5-15-2009

Application of Machine Learning Techniques for Real-time Classification of Sensor Array Data

Sichu Li
University of New Orleans

Application of Machine Learning Techniques for Real-time Classification of Sensor Array Data

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

Sichu Li

Ph.D. University of New Orleans, 1997

May, 2009

To my husband Peter Rochford, my children Ashley and Kevin Rochford,

my mother, and the memory of my father.

Acknowledgement

I would like to express my heartfelt appreciation to my thesis advisor Dr. Dongxiao Zhu for his suggestion of investigating machine learning techniques for the classification of sensor array data for my thesis topic. The author really benefited from his knowledge, advice and direction during the preparation of this thesis.

I also wish to express my appreciation to my thesis committee, Drs. Christopher Summa and Christopher Taylor, for their time in reviewing and providing constructive comments on the thesis.

In addition, I express my thanks to all of the members of the faculty in the Computer Science Department at the University of New Orleans, for their teaching in the various courses I took, where I learned and acquired my knowledge in computer science.

Finally, I would like to thank my family for their strong support during the preparation of this thesis. Without the endless encouragement from my husband, Dr. Peter Rochford, and constant inspiration from my lovely children, Ashley and Kevin Rochford, this thesis would never have been completed.

Table of Contents

List of Figures vii
List of Tables viii
Abstract ix
Chapter 1 1
Introduction 1
Chapter 2 5
Data Description 5
  2.1 Feature Vector Extraction 6
  2.2 Training and Testing Dataset 13
  2.3 Dataset for Cross-validation 13
Chapter 3 14
K-Nearest Neighbor 14
  3.1 Method 15
  3.2 Training 17
  3.3 Validation 17
  3.4 Testing 18
Chapter 4 21
Support Vector Machines 21
  4.1 Method 22
    4.1.1 Classification SVM Type 1 24
    4.1.2 Classification SVM Type 2 25
    4.1.3 Kernel Functions 25
  4.2 Training 26
  4.3 Validation 26
  4.4 Testing 27
Chapter 5 28
Classification and Regression Tree 28
  5.1 Method 29
  5.2 Training 31
  5.3 Validation 31
  5.4 Testing 32
Chapter 6 35
Random Forest 35
  6.1 Method 36
  6.2 Training 37
  6.3 Testing 38
Chapter 7 39
Naïve Bayes Classifier 39
  7.1 Method 40
  7.2 Training 42
  7.3 Testing 42
Chapter 8 43

Principal Component Regression 43
8.1 Method 44
8.2 Training 46
8.3 Validation 46
8.4 Testing 48
Chapter 9 49
Results and Discussion 49
9.1 KNN 50
9.2 SVM 54
9.3 CART 56
9.4 Random Forest 58
9.5 Naïve Bayes Classifier 60
9.6 PCR 62
9.7 Comparison 68
Chapter 10 70
Conclusion 70
References 73
Vita 75

List of Figures

Figure 1: Simulated sensor signal as a function of time for a 10-channel sensor array 9

Figure 2: Illustration of first derivative calculation within a sensor response or background window.. 10

Figure 3. Predictor values vs. predictor number (sensor channel)... 12

Figure 4: A simple KNN classification for an unknown indicated by the empty circle.............. 16

Figure 5: Overall success rate vs. k value obtained from KNN 10-fold..................................... 20

Figure 6. A simple linear SVM classification.. 22

Figure 7. An example SVM classification.. 23

Figure 8. Kernel transformation in SVM... 24

Figure 9. CART decision tree for multi-channel sensor data. 33

Figure 10. Variation of cross validation error with number of principal components for the 4 classes. 47

Figure 11. Prediction success rates for the testing dataset using KNN method. 52

Figure 12. Dendrogram of the training dataset. 53

Figure 13. Prediction success rates for the testing dataset using SVM method. 55

Figure 14. Classification success rates for the testing dataset using CART method.................. 57

Figure 15. Classification success rates using Naïve Bayes Classifer with both linear and quadratic... 61

Figure 16. Predicted Y-block values for each class... 64

Figure 17: Prediction success rates for the testing dataset based on maximum Y-Block value from PCR method. 67

Figure 18: Comparison of prediction success rates of six methods.. 69

List of Tables

Table 1: KNN 10-fold cross-validation results. 19

Table 2: Percentage success rates from KNN 10-fold cross-validation. 20

Table 3. Initial and optimal parameters for the SVM model. 27

Table 4: CART self-prediction results. 34

Table 5: Percentage success rates from CART self-prediction. 34

Table 6: Prediction results for the testing dataset using KNN method. 52

Table 7: Prediction results for the testing dataset using SVM method. 55

Table 8: Prediction results for the testing dataset using CART method. 57

Table 9. Dependence of classification success rate on the number of trees used in Random Forest method. 59

Table 10: Prediction results for the testing dataset using RF method with 50 trees. 59

Table 11: Prediction results for the testing dataset using Naive Bayes method. 61

Table 12: Prediction results for the testing dataset using PCR method. 65

Table 13: Classification of unknown samples based on their maximum Y-block value. 66

Table 14: Prediction results for the testing dataset based on maximum Y-block value from PCR method. 67

Abstract

There is a significant need to identify approaches for classifying chemical sensor array data with high success rates that would enhance sensor detection capabilities. The present study attempts to fill this need by investigating six machine learning methods to classify a dataset collected using a chemical sensor array: K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Classification and Regression Trees (CART), Random Forest (RF), Naïve Bayes Classifier (NB), and Principal Component Regression (PCR). A total of 10 predictors that are associated with the response from 10 sensor channels are used to train and test the classifiers. A training dataset of 4 classes containing 136 samples is used to build the classifiers, and a dataset of 4 classes with 56 samples is used for testing. The results generated with the six different methods are compared and discussed. The RF, CART, and KNN are found to have success rates greater than 90%, and to outperform the other methods.

Keywords: KNN, SVM, CART, Random Forest, Naïve Bayes, PCR, machine learning, classification, sensor array

Chapter 1

Introduction

An electronic nose (E-nose) is an instrument that is designed to mimic the function of the natural nose. By definition, it not only detects but also discriminates among complex chemical vapors [1], with a sensor array typically consisting of a group of non-specific chemical sensors that respond to chemical vapors. The detection, classification, and identification of a particular chemical is based on a unique combinatory response pattern from all sensors rather than the response pattern from a particular sensor. In addition to the sensor substrates, the response pattern recognition algorithm is a key component in an E-nose system that determines how well the E-nose identifies the chemicals. Improvements in the response pattern recognition algorithm can therefore provide major advances in the detection capabilities of the instrument, and drives the need to identify approaches for classifying chemical sensor array data with high success rates. In this study, we attempt to fill this need by investigating the feasibility of using machine learning techniques to classify chemical sensor array data for detection of different classes of chemical subjects.

In a chemical system, classes may differ for many reasons, including variations in sample preparation, differences in chemical compound type (aromatic, aliphatic, carbonyl, etc.), or variations in process state (startup, normal, particular faults, etc.). In this study, classes are referred to by chemical compound type (carbonyl, amine, carboxyl, and aromatic structure). Almost all chemical systems are complex because different chemical components are typically related to each other in a certain way. It is possible that all channels (sensors) could react to an individual chemical, thereby making classification of the response of a chemical sensor array non-trivial, and requiring that the classification be based on a combination of responses rather than an individual one.

A variety of methods have been developed for classifying samples based on the measured sensor response [2]. These methods fall into the two general categories of cluster analysis or unsupervised pattern recognition, and classification or supervised pattern recognition, where the former attempts to identify groups or classes without using pre-established class memberships, and the latter uses known class memberships. Classification of new unknown samples can be accomplished manually using unsupervised methods, or automatically using supervised techniques. Since our objective is to develop a classification algorithm for a real-time sensor array, the supervised approach is of interest because it can be implemented to provide classification results online.

To classify multi-channel sensor data, typically a Principal Component Analysis (PCA)-based method is used to decompose the covariance matrix among different sensor channels into orthogonal Principal Components (PCs), and then a certain discriminating approach is used for classification. Chemometric methods have also been widely exploited to classify chemicals for sensor arrays composed of multiple channels (a group of sensors) [3, 4]. Typical applications of chemometric methods employ the development of quantitative structure activity relationships or the evaluation of analytical–chemical data to perform the classification. In addition, a variety of statistical methods have been adapted into chemometrics for classification to solve chemical-related problems such as the K-nearest neighbor, PCA, and the Partial Least Squares Discriminant Analysis (PLS-DA) [5, 6]. The success rates using these approaches are rarely high enough to satisfy instrument detection requirements, and the classification model needs to be recalibrated often. This drives us to explore alternative approaches to improve classification.

In this study, we explore using supervised machine learning techniques to improve situation awareness in order to optimize sensor performance. Machine learning is an effective and practical technique for discovering relations and extracting knowledge because they are able to map inputs to desired outputs in a robust manner. Since detection targets are known in most cases, we can obtain a training dataset by testing the sensor array with standard chemical samples. In addition, it is possible to collect the training dataset under a variety of operational conditions to ensure that the dataset is representative. A variety of machine learning methods are considered here to evaluate their potential benefits for classification of multi-channel sensor data because the distinct mechanisms each employs.

The objective of this study is to compare the predictive accuracy of six of the more popular classifiers used in the scientific community on a multi-channel chemical sensor array dataset: K-Nearest Neighbor (KNN), Support Vector machine (SVM), Classification and Regression Trees (CART), Random Forests (RF), Naïve Bayes Classifier (NB), and Principal Component Regression (PCR). The sensor dataset is constructed from 4 classes of 48 data samples, with 34 samples selected from each class to form a training dataset of 136 (34×4) data samples. The remaining samples are used to form a testing dataset of 56 (14×4) data samples. Since the individual sensors (channels) of the sensor array respond to chemical stimuli differently in their signal change (increase or decrease, intensity of the change), a predictor is assigned to each channel to reflect its contribution to a combined response pattern of the sensor array. Therefore, 10 predictors (variables) are used in training and testing the classifiers.

In the remainder of this thesis, we first describe the nature of the dataset and the features captured within it (Chapter 2). We then present six classifier methods (Chapters 3 to 8), providing a brief description of each method, the training of each model, and its validation and testing. Finally, we present and discuss the prediction results generated using each classifier (Chapter 9), and provide a conclusion based on the results and observations (Chapter 10).

Chapter 2

Data Description

The multi-channel sensor dataset used for this study was collected by challenging a sensor array with 4 groups of chemicals with different molecular structures (carbonyl, amine, carboxyl, and aromatic structure). Each data sample is a combined response from 10 different types of chemical sensors on a sensor array, where each chemical sensor is referred to as a sensor channel. Each group of chemicals was tested for three concentration levels (high, medium, and low), and the data collected from challenges with the same group of chemicals forms a class. The steps carried out to construct the 4 classes within the dataset used for this study are described next.

2.1 Feature Vector Extraction

A response to a chemical stimulus from a sensor channel is triggered by a chemical reaction between the sensing element and the chemical. The reaction results in a change in a particular property (ΔP), and this change typically depends on the chemical concentration (C). The relationship between ΔP and C can be non-linear over a long time window due to the change in sensor sensitivity over time. However, for online classification, the time period over which the sensor data is collected is invariably short. For example, for this sensor array a time window covered by 3 data points after the start of sampling. The sensor response is typically assumed to have a linear relationship with the concentration of the collected chemical, as expressed by the relation

$$\Delta P = \alpha\, C$$

where α is a parameter that is highly associated with the sensor channel behavior. For a single sensor channel, the sensitivity and chemical collection efficiency determine its parameter α. Therefore, α can be used as a marker to indicate the behavior of each sensor channel. Furthermore, since all sensor channels in the sensor array are subject to the same concentration of a chemical stimulus, ΔP can then be used to indicate the sensor channel behavior. The question is how to express the property change ΔP. Since every measured signal includes not only a signal of the property of interest but also contributions from the background (due to certain cross-reactions with chemicals residing in the environment) and some system noise, ΔP must be calculated in a manner to minimize such effects. To reduce noise, ΔP is evaluated as a rate of change over a given time window $(\Delta P/\Delta t)_r$. To remove the background contribution, the rate of change $(\Delta P/\Delta t)_b$ is calculated within a time window just before the sensor experiences the chemical stimuli, and this value is then subtracted from the rate of change (($\Delta P/\Delta t)_r$ within the same length time window immediately after. Since the sensor array was developed to meet a system requirement of classifying the chemical stimuli within a short time window, for which its sampling capability could only provide 3 data points, the time window used to evaluate the rate of change in sensor signal was limited to 3 data points.

Figure 1 shows an example of simulated data from the 10 sensor channels before and after a chemical challenge occurs at the point indicated by “Start challenge”. The data reveals that each sensor channel responds to the challenge with a different rate of signal change. For some sensor channels there is a noticeable baseline drifting prior to the challenge, while for some sensor channels the response is more rapid than others. To adequately capture this signal change over time, a rate is calculated for each sensor channel s_n as the difference between the instantaneous rate and a baseline signal drift rate

$$x_n = \left(\frac{\Delta s_n}{\Delta t}\right)_r - \left(\frac{\Delta s_n}{\Delta t}\right)_b$$

where n is the sensor channel number (1 to 10). As shown in Figure 2, $\left(\frac{\Delta s_n}{\Delta t}\right)_r$ is the signal change rate after the start of the challenge, and $\left(\frac{\Delta s_n}{\Delta t}\right)_b$ is the baseline signal drifting rate immediately before challenge. The time window (Δt) used for calculating the signal change rate is a 3 data point period for the reason given above. The actual values of the signal change rate are evaluated from a least-squares fit of the 3 data point values. The resultant linear function is

$$s = a\,t + c$$

from which one obtains the signal change rate $\left(\frac{\Delta s_n}{\Delta t}\right)_r = a$ and $\left(\frac{\Delta s_n}{\Delta t}\right)_b = a$ for data points in the Δt_r and Δt_b time windows, respectively. These signal change rates for the sensor channels are used to define 10 feature element values (predictors) that form the training and testing datasets to which the classification models described further below are applied. For the sensor channels that respond to the challenge more quickly, the time window is from the first to third data point reading after a challenge. For those that have a slower response, Δt starts from the second to fourth data reading after a challenge, i.e. one data point reading delay. Thus each data sample is converted to a vector:

$$\vec{x} = (x_1, x_2, x_3, \ldots x_{10})$$

where $x_1, x_2, x_3, \ldots x_{10}$ are the 10 predictors associated with the 10 sensor channels.

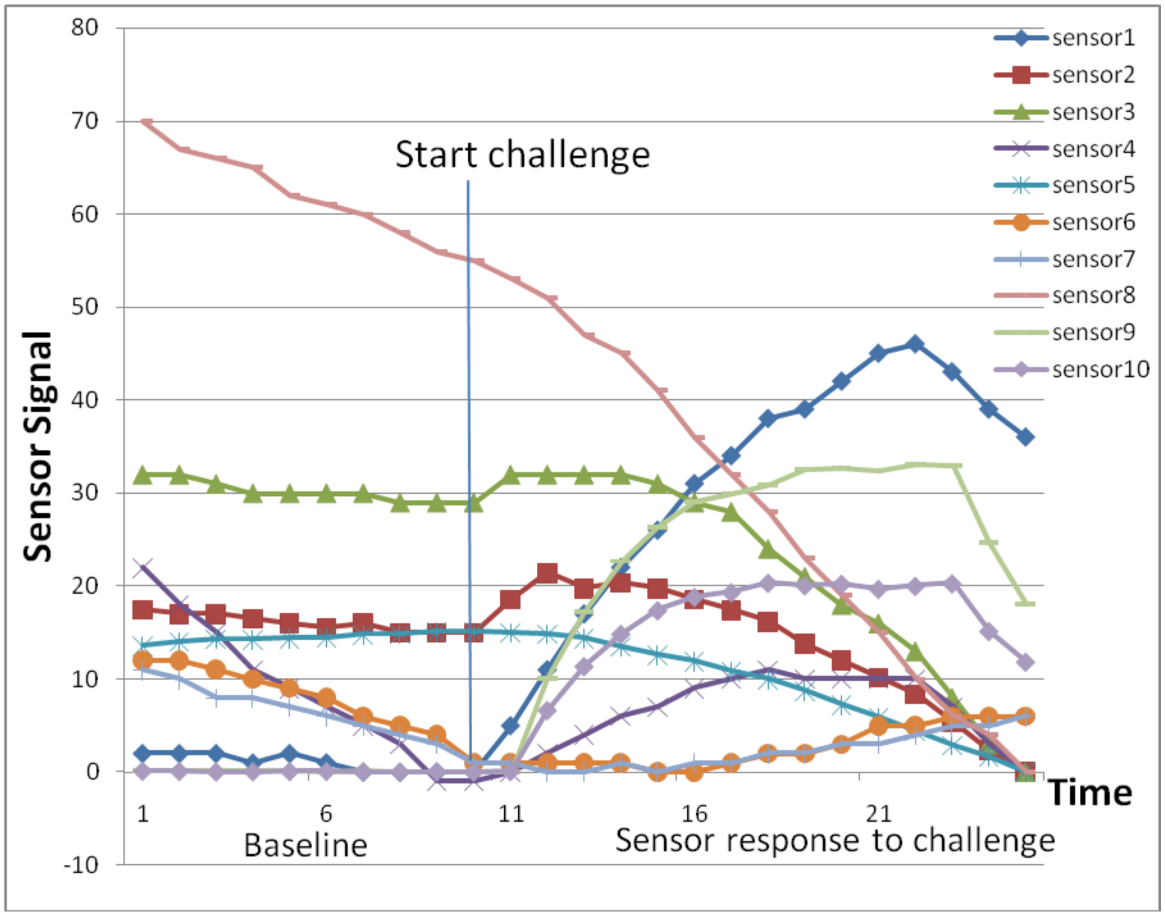


Figure 1: Simulated sensor signal as a function of time for a 10-channel sensor array before and after a chemical challenge.

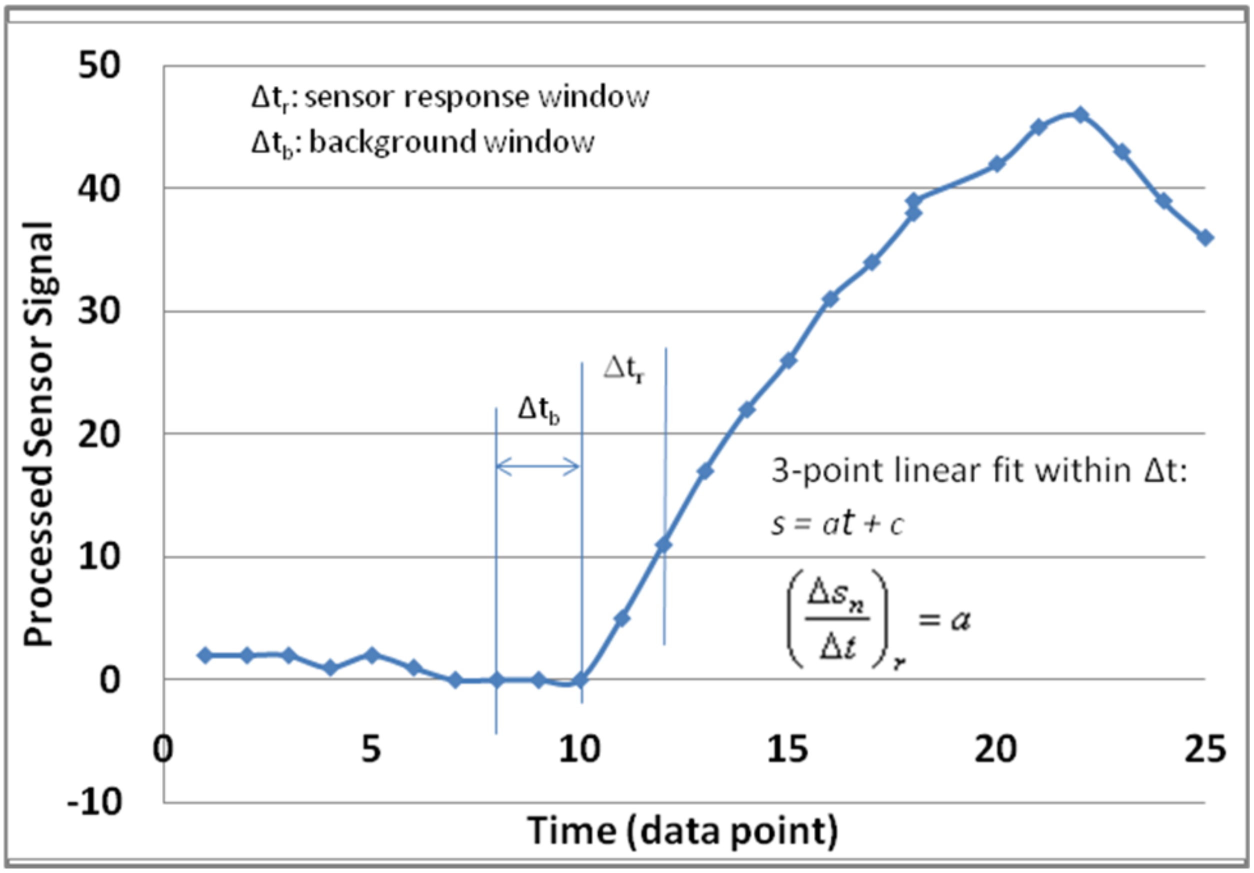


Figure 2: Illustration of first derivative calculation within a sensor response or background window.

Plots of predictor value versus predictor number (sensor channel number) for each training data sample for 4 classes are shown in Figure 3. From the general variation in the curves it can be seen that the features of classes 3 and 4 are more distinct than the others, while the features of classes 1 and 2 exhibit certain cross-reactivity between each other and with class 3. While quantities other than sensor signal change rate might be used for the feature vector, as stated above, it is employed here because it provides the most direct indication of sensor behavior that can be employed to reduce noise. Moreover, a least-squares fit provides the best statistical means for estimating the change rate.

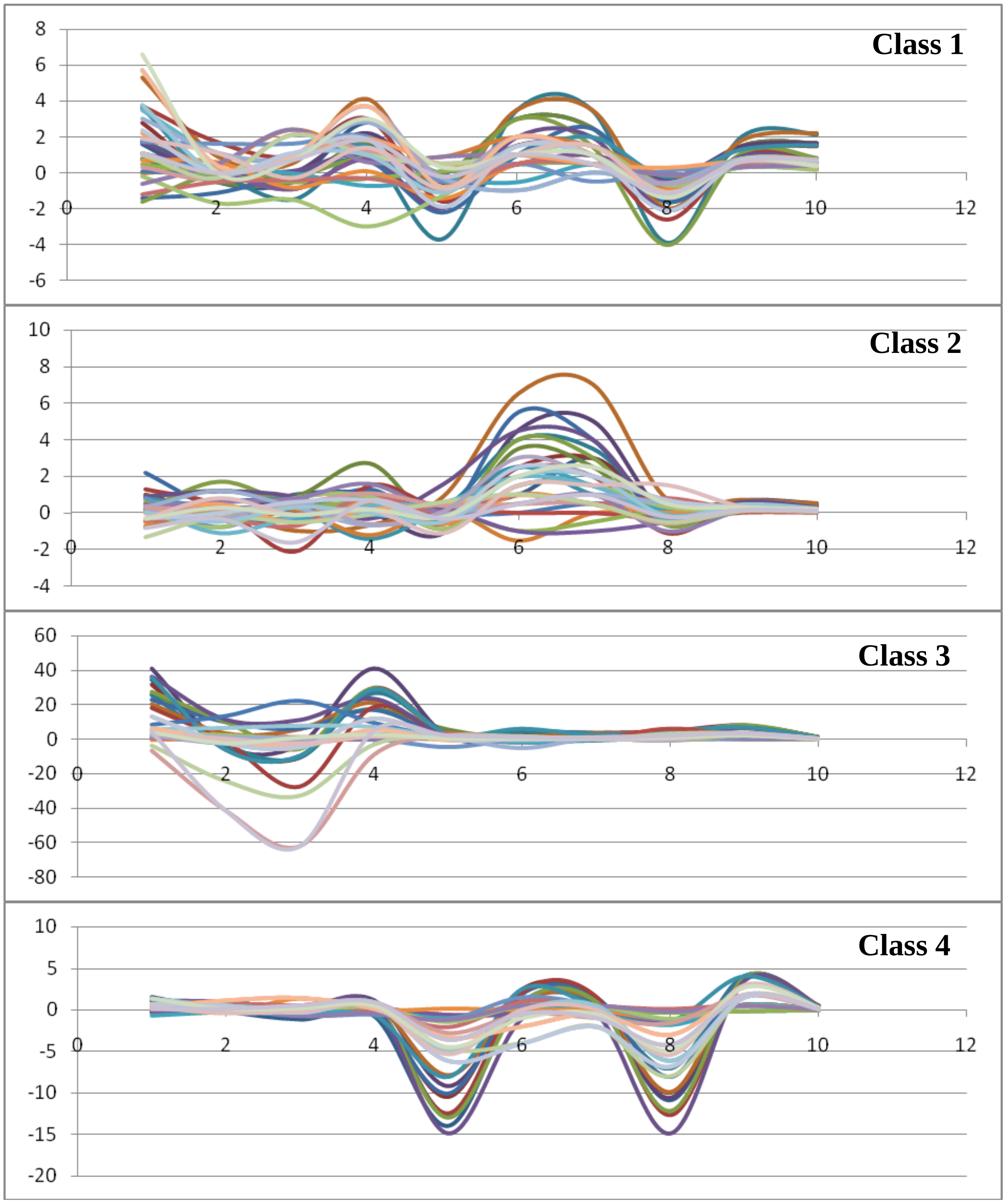


Figure 3. Predictor values vs. predictor number (sensor channel).

X axis: predictor number (sensor channel number)
Y axis: predictor value

2.2 Training and Testing Dataset

In this study there are 4 classes of 48 data samples, with 34 samples randomly selected from each class to form a training dataset of 136 (34×4) data samples. The remaining samples are used to form a testing dataset of 56 (14×4) data samples. Because the sensor behavior depends on environmental operational conditions such as temperature and humidity, and the 48 data samples in each class were collected from 4 trials, each with different operational conditions, the 34 data samples in each class were randomly selected from the 48 data samples to ensure the training dataset captured the variability in conditions. Furthermore, since the 48 data samples in each class have equal numbers of 16 samples with respect to high, medium, and low challenge concentrations, the number of data samples randomly selected for each of the three concentration levels is nearly equal in both the training and testing datasets. This ensures a fair representation from each concentration level within a class. Figure 3 provides predictor values vs. predictor numbers for the training dataset.

2.3 Dataset for Cross-validation

In constructing some machine learning models for prediction, we performed a cross validation to obtain the optimal model parameter values, e.g. K value in KNN, Cross validation is a method to estimate the error rate in an efficient and unbiased way. The procedure works as follows: the dataset is first divided into k sub-samples ($k \leq 10$) (in our experiments $k = 10$ for each class). A single sub-sample is then chosen as testing data, and the remaining $(k - 1)$ subsamples are used as training data. The procedure is repeated k (10) times, in which each of the k subsamples is used exactly once as the testing data. All testing results, 34 predictions for each class, are averaged and a single estimation is provided [2].

Chapter 3

K-Nearest Neighbor

3.1 Method

The k-nearest neighbor (KNN) method [7] is one of the most popular algorithms to implement because it is simple, powerful, yet model free. It functions on the intuitive idea that close objects are more likely to be in the same class. The method finds k known observations x in a training dataset that are closest to the unknown test observation y to predict the classification of the latter (Figure 4), where k is a positive integer that is typically small. The class of y is predicted based on the majority vote of the classes of these k neighbors, hence the name *k*-Nearest Neighbor. In the simplest case when k = 1 (nearest neighbor), y is simply assigned to the class of its nearest neighbor x. Closeness implies a metric for the distance $D(x, y)$ between x and y that must be defined, such that the smaller the distance, the closer x is to y. One of the most popular choices for the metric is Euclidean distance, while other choices are Euclidean squared, City-block, and Chebychev:

$$D(x,y)=\begin{cases}\sqrt{(x-y)^2} & Euclidian\\ (x-y)^2 & Euclidian\,squared\\ abs(x-y) & City-block\\ \max(|x-y|) & Chebychev\end{cases}$$

where the difference between x and y is in terms of a difference in vectors. The KNN method relies on having a set of observations for which the outcome is known, i.e. a set of independent values labeled by a set of dependent outcomes. (The independent and dependent variables can be either continuous or categorical). For continuous dependent variables, the task is regression; otherwise it is a classification. For the application being investigated here the KNN method is being applied as a classification.

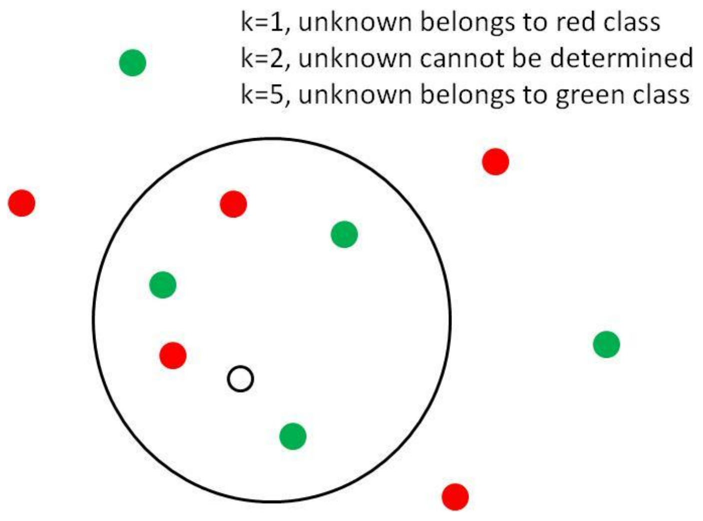

k=1, unknown belongs to red class
k=2, unknown cannot be determined
k=5, unknown belongs to green class

Figure 4: A simple KNN classification for an unknown indicated by the empty circle.

The choice of *k* is essential in building the KNN model as it can strongly influence the quality of predictions. One appropriate way to look at the number of nearest neighbors *k* is to think of it as a smoothing parameter. For any given problem, a small value of *k* will lead to a large variance in predictions. Alternatively, setting *k* to a large value may lead to a large model bias. Thus, *k* should be set to a value large enough to minimize the probability of misclassification and small enough (with respect to the number of observations in the training dataset) so that the *k* nearest observations are close enough to the unknown test observation . Thus, like any smoothing parameter, there is an optimal value for *k* that achieves the right trade off between the bias and the variance of the model.

The value of k can be found using a cross-validation algorithm. It entails dividing the data sample into a number of sub-samples. For a fixed value of *k*, the KNN model is applied to make predictions on one sub-sample (using the remaining subsamples as the prototype examples) and evaluate the error. For classification, the error is most conveniently defined as the accuracy (the percentage of correctly classified observations). This process is then successively applied to all possible choices of sub-samples. After cycling through all the sub-samples, the computed errors are averaged to determine how well the model predicts the known test observations. The above steps are then repeated for various *k* and the value achieving the lowest error (or the highest classification accuracy) is then selected as the optimal value for *k* (optimal in a cross-validation sense). Note that cross-validation is computationally expensive.

3.2 Training

The PLS_Toolbox (Version 4.2) developed by Eigenvector Research Inc. for multivariate analysis of chemometrics within the Matlab™ computational environment [8] was used to perform the KNN training. The "knn" function within PLS_Toolbox was employed for both cross-validation and the predicted classification. The feature vectors in the training and testing datasets were supplied to the function as a N×10 float array of data, where each row represented a single feature vector. The class values for both datasets were provided as a N×1 integer array, with each row being the class value (1 through 4) of the corresponding feature vector in the N×10 float array of data. Autoscaling was applied to both the training and testing feature vectors by centering the columns of the feature vectors to zero mean and scaling them to unit variance, and the closest neighbor was assigned to the predicted class if there was no majority vote amongst the nearest neighbors.

3.3 Validation

The cross-validation to determine the optimal choice of k nearest neighbors was accomplished by excluding a subset of the training dataset of N (N = 10 in this study) subsets of vectors, constructing the KNN model with the remaining N-1 subsets of vectors, and then using the constructed model to predict the class of the excluded vector. This process was repeated for all feature vectors in the training set, accumulating statistics on the number of successful predictions by comparing the predicted class of the excluded vector against its known class. The cross validation was performed for odd values k = 1 through 9. The results are listed in Table 1, and the success rates were calculated and shown in Table 2. These results indicate the constructed model has a very high performance for prediction within the training dataset, with a constant success rate for the class 1 and 2 features, small variability for class 4, and moderate variability for class 3. As shown in Figure 5, the highest success rate is obtained when using k=1, i.e. the nearest neighbor. The average success rate for all classes decreases for k=1 to 5 and then slightly increases to a constant value thereafter, indicating that increasing the number of nearest neighbors saturates the voted closest class beyond k =7. Therefore, k=1 is used for the KNN model prediction of the sensor array data.

3.4 Testing

The KNN method was tested by using the “knn” function to build the model with the training dataset and k=1 as input along with the option of autoscaling. The results are shown and discussed in Chapter 9.

Table 1: KNN 10-fold cross-validation results.

k = 1	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	31	0	0	0	94.1
P - Class 2	2	32	0	1	
P - Class 3	1	1	32	0	
P - Class 4	0	1	2	33	

k = 3	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	31	0	0	0	93.4
P - Class 2	2	32	2	0	
P - Class 3	0	1	30	0	
P - Class 4	1	1	2	34	

k = 5	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	31	0	0	0	91.2
P - Class 2	2	32	5	1	
P - Class 3	0	1	28	0	
P - Class 4	1	1	1	33	

k = 7	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	31	0	0	0	92.6
P - Class 2	2	32	4	0	
P - Class 3	0	0	29	0	
P - Class 4	1	2	1	34	

k = 9	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	31	0	0	0	92.6
P - Class 2	2	32	3	0	
P - Class 3	1	0	29	0	
P - Class 4	0	2	2	34	

T - Class: Truth class ID
P - Class: Predicted class ID

Table 2: Percentage success rates from KNN 10-fold cross-validation.

k	Class 1	Class 2	Class 3	Class 4	Overall
1	91.2	94.1	94.1	97.1	94.1
3	91.2	94.1	88.2	100.0	93.4
5	91.2	94.1	82.4	97.1	91.2
7	91.2	94.1	85.3	100.0	92.6
9	91.2	94.1	85.3	100.0	92.6

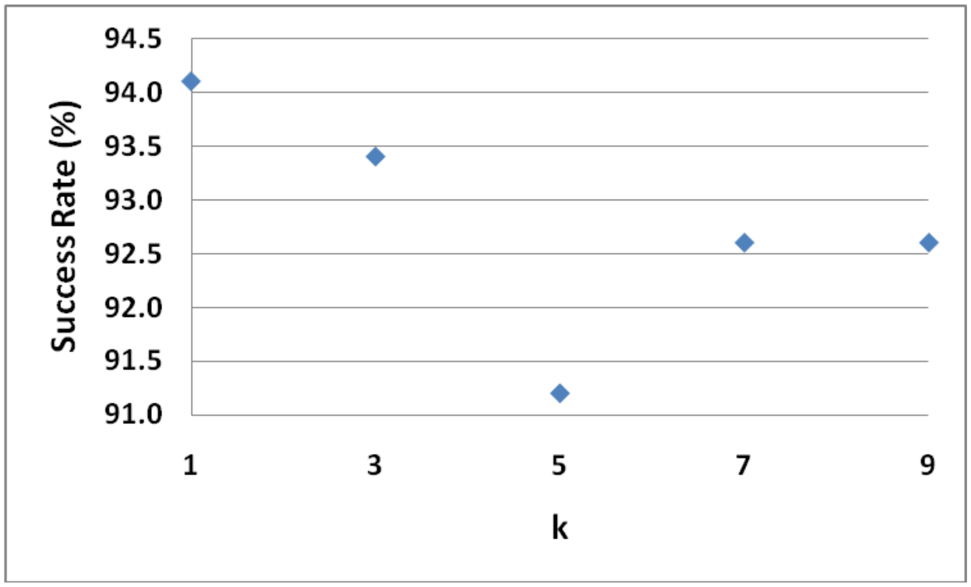


Figure 5: Overall success rate vs. k value obtained from KNN 10-fold cross-validation.

Chapter 4

Support Vector Machines

4.1 Method

Support Vector Machines (SVM) is a powerful technique for classification problems [9, 10], and is one of the most popular classifiers these days. It is based on the concept of decision planes that define decision boundaries separating sets of objects having different class memberships. It is primarily a classifier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. Because of the nature of the feature space in which these boundaries are found, SVMs can exhibit a large degree of flexibility in handling classification and regression tasks of varied complexity.

The goal of SVM is to construct a separating hyperplane that is maximally distant from different classes of the training data. To illustrate, an example is shown in Figure 6 of objects that belong to either class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

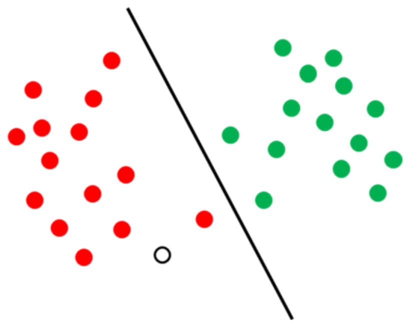


Figure 6. A simple linear SVM classification.

This is a classic example of a linear classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. However, most classification tasks are not so simple, and require more complex discriminating functions to make an optimal separation, i.e., correctly classify new objects on the basis of the examples that are available. This is illustrated by the example in Figure 7 where a full separation of the GREEN and RED objects would require a curve.

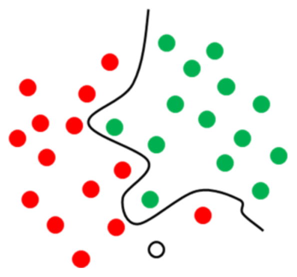


Figure 7. An example SVM classification.

The basic idea behind SVM is to rearrange the original objects using a set of mathematical functions (kernels) into a linearly separable arrangement as illustrated Figure 8. As a result of the transformation, the task of separation is reduced to finding an optimal line that can separate the GREEN and the RED objects on the right side of Figure 8 rather than constructing the complex curve as appears on the left side. As shown in Figure 8 the points lying on the boundaries are called support vectors, and the middle of the margin is the optimal separating hyperplane that maximizes the margin of separation. Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are referred to as hyperplane classifiers.

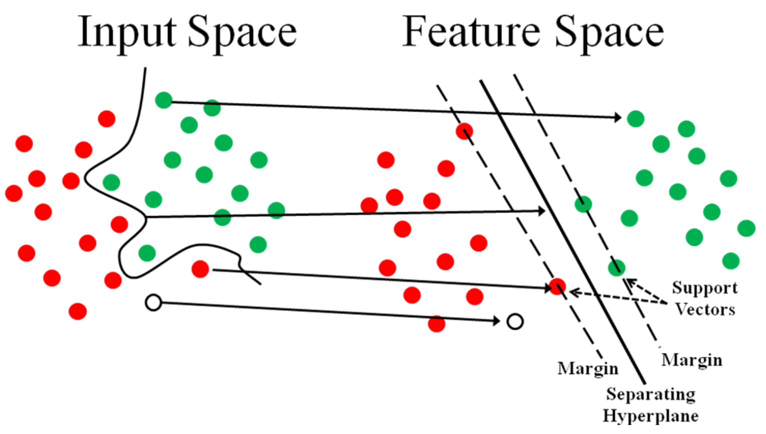


Figure 8. Kernel transformation in SVM.

For categorical variables a dummy variable is created with case values of either 0 or 1. Thus, a categorical dependent variable consisting of three levels, say (A, B, C), is represented by a set of three dummy variables:

$$A:\{1\ 0\ 0\}, B:\{0\ 1\ 0\}, C:\{0\ 0\ 1\}$$

To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize an error function. There are two distinct groups for classification SVM models according to the form of the error function.

4.1.1 Classification SVM Type 1

For this type of SVM, training involves the minimization of the error function

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i ,$$

subject to the constraints

$$y_i(w^T\phi(\vec{x}) + b) \geq 1 - \xi \text{ and } \xi_i \geq 0,\ i = 1,\dots,N ,$$

where C is the capacity constant, w is the vector of coefficients, b a constant and the ξ_i are parameters for handling non-separable data (inputs). The index i labels the N training cases. Note that $y \in \pm 1$ is the class labels and $\bar{x}$ is the vector of independent variables. The kernel ϕ is used to transform data from the input (independent) to the feature space (see below). The vector w is normal and perpendicular to the hyperplane. The parameter $b/|w|$ determines the offset of the hyperplane from the origin along the normal vector w. It should be noted that the larger the C, the more the error is penalized. Thus, C should be chosen with care to avoid over fitting.

4.1.2 Classification SVM Type 2

In contrast to Classification SVM Type 1, the Classification SVM Type 2 model minimizes the error function

$$\frac{1}{2} w^T w - \nu\rho + \frac{1}{N}\sum_{i=1}^{N} \xi_i \ ,$$

subject to the constraints

$$(w^T \phi(\vec{x}) + b) \geq \rho - \xi_i \text{ and } \xi_i \geq 0, i = 1,\ldots,N \text{ ,and } \rho \geq 0 .$$

4.1.3 Kernel functions

There are a number of kernels that can be used, and these include linear, polynomial, radial basis function (RBF), and sigmoid:

$$\phi(\vec{x},\vec{x}') = \begin{cases} \vec{x}\cdot\vec{x}' & \mathit{Linear} \\ (\gamma\vec{x}\cdot\vec{x}' + c)^n & \mathit{Polynomial} \\ \exp(-|\vec{x}-\vec{x}'|^2/\sigma^2),\ \sigma^2 > 0 & \mathit{RBF} \\ \tanh(\gamma\vec{x}\cdot\vec{x}' + c) & \mathit{Sigmoid} \end{cases}$$

The RBF is by far the most popular choice of kernel types used in SVMs. This is mainly because of their localized and finite responses across the entire range of the real x-axis. Although SVMs are very powerful and commonly used in classification, they suffer from several drawbacks. They require high computations to train the data, and they are sensitive to noisy data thereby being prone to over fitting.

4.2 Training

The Least Squares Support Vector Machine (LS-SVM) toolbox (Version 1.5) provided by the Katholieke Universiteit Leuven, Belgium [11-13] was used to build a SVM for the multi-sensor data. To set up the model for classification, the class values of the training dataset were first organized into a N×4 Y-block with values of 0 or 1, where the first column indicated the feature vectors in the N×10 float array belonging to class 1, the second column to class 2, etc. A value of 1 was used to identify the given feature vector as belonging to the class, and a value of 0 if it was not a member of the class. A RBF was chosen as the kernel for the model because of its localized and finite response across the entire range of the real x-axis. The LS-SVM toolbox uses Type 1 classification by default.

4.3 Validation

The "tunelssvm" function was used to determine the optimal hyperparameters for the model: the values for the regularization parameter (γ), determining the trade-off between the fitting error minimization and smoothness, and the RBF kernel parameter (σ^2). The tuning was performed for classification ("type" of "c") using the training dataset, with independent γ and σ^2 parameter values for each of the 4 columns of the Y-block. The tuning function uses L-fold cross validation provided by the "crossvalidate" function to determine the optimal parameters. In this type of cross validation the data is permutated randomly once, then divided into L (by default 10) disjoined sets. In the i-th (i=1, ..., L) iteration, the i-th set is used to estimate the performance (validation set) of the model trained on the other L-1 sets (training set). In the final step the L different estimates of the performance are averaged (mean). The assumption is made that the input data are distributed independent and identically over the input space. The tuning function yielded the optimal values shown in Table 3 for the indicated initial values of γ_i and σ_i^2. The "trainlssvm" function was then employed to train the SVM for classification using the training dataset. The support values (α) and bias (b) obtained were then used with the "simlssvm" function to predict the class values for the feature vectors of the training dataset.

Table 3. Initial and optimal parameters for the SVM model.

Parameter	Class 1	Class 2	Class 3	Class 4
γ_i	1	1	1	1
σ_i^2	1	1	1	1
γ	69.1073	38.3934	32.3379	74.3043
σ^2	1.5675	1.3146	3.4980	7.0838

4.4 Testing

The SVM model was tested by using the "simlssvm" function to make predictions for the testing dataset. The training dataset was input along with the optimal γ and σ^2 parameter values, specification of a RBF kernel, the support values (α) and bias (b) obtained from the tuning, and the feature vectors of the testing dataset. The results obtained are shown and discussed in Chapter 9.

Chapter 5

Classification and Regression Tree

5.1 Method

CART is a well-known method for constructing a classification tree from data [14]. It is a model that describes the conditional distribution of y given x. It consists of two components: a tree T with b terminal nodes, and a parameter vector **p**= ($p_1, p_2, \ldots, p_b$) where p_i is associated with the ith terminal node. Each terminal node of the tree corresponds to a distinct region for y, which is discrete for the case of classification, or continuous in the case of regression. In its most basic form, the classification tree is a binary tree, where each leaf of a node corresponds to a range of values of a domain variable (attribute). The partition is determined by splitting rules associated with the internal nodes of the binary tree. Should the splitting variable be continuous, a splitting rule in the form $x_i \in C$ and $x_i \notin C$ is assigned to the left and the right of the split node respectively, where $\vec{x} = (x_1, x_2, \ldots, x_m)$ and C is a classification category. However, should the splitting variable be discrete, a splitting rule in the form $x_i \leq s$ and $x_i > s$ is assigned to the right and the left of the splitting node respectively.

The predictor at each node is determined by searching through the variables x_1 to x_m one by one. For each x_i variable, a splitting rule is applied starting from an initial estimate for the s_i parameter value, and the goodness of split determined by evaluating an impurity function $i(C)$ on the outcome. The s_i parameter values are then recursively adjusted until the impurity function is separately minimized for each x_i variable, thereby defining the best split for each x_i variable. The algorithm then compares the m best single variable splits and selects the best from amongst these as the splitting rule for the node. The Gini index of diversity is typically employed for the impurity function

$$i(C) = \sum_{i \neq j} p(i \mid C) p(j \mid C)$$

where $p(j \mid C)$ is the proportion of the cases $x_i \in C$ belonging to class j.

The CART is applied by using known attribute values to traverse a path down the tree to a leaf node. In constructing a classification tree, the typical goal is to build the single tree that maximizes expected classification accuracy on new cases. Typically, a one-step look ahead is used in constructing branch points. In an attempt to avoid over fitting, trees often are pruned by collapsing subtrees into leaves.

Generally, CART analysis consists of three basic steps. The first step consists of tree building, during which a tree is built using recursive splitting of nodes. After a large tree is identified, the second stage of the CART methodology uses a pruning procedure that incorporates a minimal cost-complexity measure. The result of the pruning procedure is a nested subset of trees starting from the largest tree grown and continuing the process until only one node of the tree remains. A testing sample is used to provide estimates of future classification errors for each sub-tree. The last stage of the methodology is to select the optimal tree, which corresponds to a tree yielding the lowest testing set error rate.

CART is flexible in practice in the sense that it can easily model nonlinear or non-smooth relationships. It has the ability of interpreting interactions among predictors and has great interpretability due to its binary structure. However, CART has several drawbacks, one of which is to overestimate the data.

5.2 Training

The Statistics Toolbox (Version 6.1) provided by MathWorks as part of the Matlab™ (Release 2007b) computational environment [8] was used to build a CART for the multi-sensor data. The “classregtree” function within the Statistics Toolbox was employed to build a decision tree for predicting the class of the multi-channel sensor response as a function of the predictors derived from the data in the ten sensor channels. The feature vectors for the training dataset were supplied to the function in the same format as for the KNN model above, while the class values were input as a N×1 array of string values. Predictor names of “s1”, “s2”, through to “s10” were also provided to the function for use as labels when producing a graphical representation of the tree. The split criterion applied to determine which predictor to use to split nodes was Gini's diversity index (the default). Construction of a CART using the training dataset, with no application of additional features such as tree pruning and splitting, yielded the decision tree shown in Figure 9.

5.3 Validation

To assess the quality of the constructed model, the CART was applied to the training dataset used for its construction, and a success rate determined by comparing the class predicted for each feature vector against its known class. This entailed calling the “eval” Matlab function with the feature vectors from the training dataset and the decision tree generated from the “classregtree” function. The self-prediction results are listed in Table 4, and the number of successful predictions with respect to the total number of feature vectors for each class was computed as a percentage (Table 5). The very high percentage values for each of the classes with an average of 95.6% indicates the decision tree is well constructed with no need for further modifications.

5.4 Testing

The CART was tested by using the “eval” Matlab function to make predictions with the feature vectors from the testing dataset and the decision tree generated from the “classregtree” function. The number of successful predictions with respect to the total number of feature vectors for each class was computed as a percentage, and the results obtained are shown and discussed in Chapter 9.

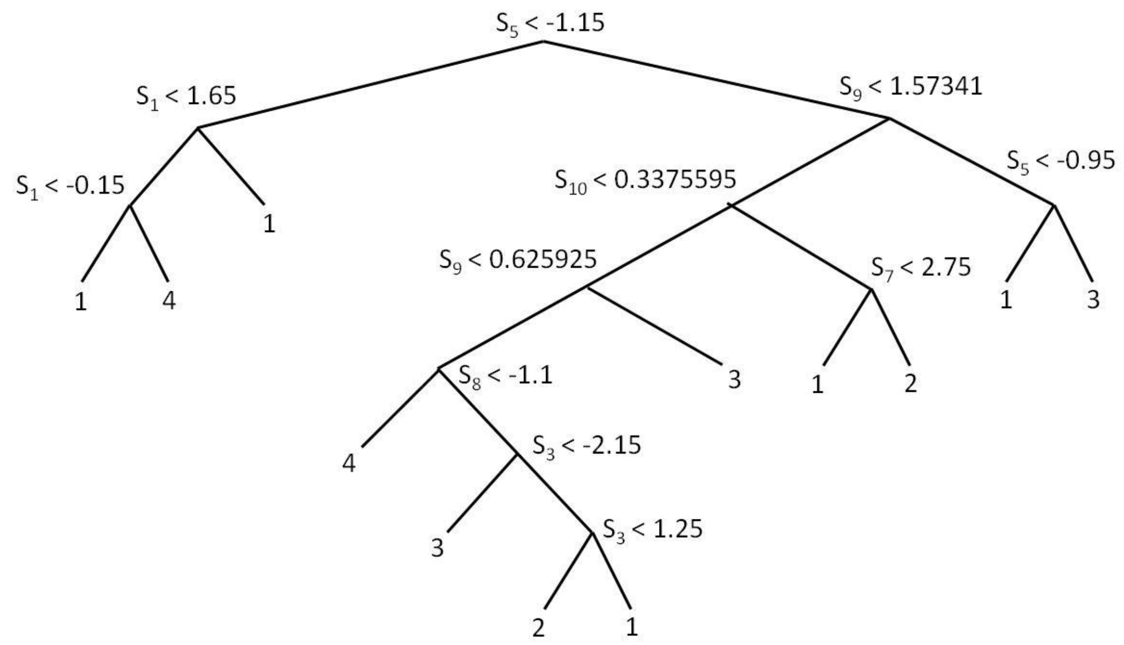


Figure 9. CART decision tree for multi-channel sensor data.

Table 4: CART self-prediction results.

CART	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	32	0	0	0	95.6
P - Class 2	1	33	0	0	
P - Class 3	0	0	33	0	
P - Class 4	1	1	1	32	

T - Class: Truth class ID

P - Class: Predicted class ID

Table 5: Percentage success rates from CART self-prediction.

Class 1	Class 2	Class 3	Class 4	Average
94.1	97.1	97.1	94.1	95.6

Chapter 6

Random Forest

6.1 Method

A Random Forest (RF) is a classifier founded on the premise that the average of predictions obtained from a statistical ensemble of classifiers will produce a more accurate prediction than one obtained from a well-constructed individual classifier [15]. The method consists of constructing many decision trees (e.g. CART), where each tree depends on the values of a random vector sampled independently with replication (bootstrap) and with the same distribution for all trees [16]. All trees are run on the sample to be classified, and the mode of the statistical distribution of classes output is chosen as the class assigned for the sample. The algorithm for inducing a random forest was developed by Leo Breiman and Adele Cutler, and "Random Forests" is their trademark [17]. Random forests have been shown to give excellent performance on a number of practical problems and are among the most accurate general-purpose classifiers available (see for example [16]). They work fast, generally exhibit a substantial performance improvement over single tree classifiers such as CART, and yield generalization error rates that compare favorably to the best statistical and machine learning methods.

For the Random Forest classifier, the algorithm proceeds as follows:

1) Choose the number of trees N to grow.
2) If there are M input variables, select a number $m \ll M$ such that at each node, m variables (features) are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3) Grow the N trees. When growing each tree:
   (a) Construct a bootstrap sample of size n sampled from the dataset *with replacement* and grow a tree from this bootstrap sample.

   (b) When growing a tree at each node select m variables at random and use them to find the best split.

   (c) Grow the tree to a maximal extent. There is no pruning.

4) To classify a testing point X, collect votes from every tree in the forest and then use majority voting to decide on the class label.

In random forests, there is no need for cross-validation to get an unbiased estimate of the test set error because it is estimated internally using an out-of-bag (OOB) error estimate during the forest building process. In the OOB method, each tree is constructed using a different bootstrap sample from the original data, where about one-third of the cases are left out of the bootstrap sample and not used in the construction of the k^{th} tree. Each case excluded from the construction is then run through the k^{th} tree to obtain a classification, thereby obtaining a test set classification in about one-third of the trees. At the conclusion of the random forest construction, the proportion of times the predicted class is not equal to the true class is averaged over all the cases, and this average value is defined as the OOB error estimate.

Random forests can handle large numbers of variables in a dataset. In addition, they can estimate missing data well. A major drawback of random forests is the lack of reproducibility, as the process of building the forest is random. Further, interpreting the final model and subsequent results is difficult, as it contains many independent decision trees.

6.2 Training

A collection of Matlab™ functions written by Xue-wen Chen and Jong Cheol Jeong at The University of Kansas for prediction of protein interaction sites [18, 19] were used to construct the random forest model. The functions build upon the “classregtree” function within the Statistics Toolbox, and construct decision trees for predicting the class of the multi-channel sensor response as for CART above. The “randomforest” function was employed to build a forest of decision trees for a varying number of trees to determine the change in class predictions with forest size. All available samples and envolved features (10 feature element values) of the training and testing datasets were used. The RF was constructed with a regenerated data set generated by bootstrap with replacement. The training and testing datasets were input to the “randomforest” function in the same manner as described for CART above.

6.3 Testing

The RF was tested by using the “randomforest” function to make predictions with the feature vectors from the testing dataset. The training and testing datasets were input along with the number of trees, the number of involved features specified as 10, and the number of samples set at the full quantity of 134 in the training dataset. The number of successful predictions with respect to the total number of feature vectors for each class was computed as a percentage, and the results obtained for varying numbers of trees are shown and discussed in Chapter 9.

Chapter 7

Naïve Bayes Classifier

7.1 Method

The Naive Bayes Classifier technique [20] is based on the Bayesian theorem, where the latter postulates that a probability can be assigned to a hypothesis being rejected or accepted. It invokes the simple assumption that the independent variables are statistically independent. Given its simplicity, the Naive Bayes Model can provide effective classification tools that are easy to use and interpret, and can often outperform more sophisticated classification methods. Naive Bayes classifiers are particularly well suited for problems with inputs having a high number of dimensions, and can handle an arbitrary number of independent variables whether continuous or categorical.

Given a set of variables, $\mathrm{X} = \{\mathrm{x}_1, \mathrm{x}_2, ..., \mathrm{x}_d\}$, the probability is constructed for an event C_j occurring among a set of possible outcomes $C = \{C_1, C_2, \ldots, C_d\}$. In the current application, X is the vector of predictors and C is the set of categorical levels present in the dependent variable. Using Bayes' rule:

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d \mid C_j) p(C_j)$$

where $p(C_j \mid x_1, x_2, \ldots, x_d)$ is the posterior probability of class membership, i.e., the probability that X belongs to C_j, $p(x_1, x_2, \ldots, x_d \mid C_j)$ is the likelihood of X occurring among the possibilities within only class C_j. and $p(C_j)$ of class C_j occurring among the possibilities. Since Naive Bayes assumes the conditional probabilities of the independent variables are statistically independent, the likelihood is decomposed into a product of terms

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j),$$

and the posterior probability is rewritten as

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(C_j)\prod_{k=1}^{d} p(x_k \mid C_j).$$

Using Bayes' rule above, the new case X is labeled with the class level C_j that achieves the highest posterior probability.

Although it is not always accurate to assume the predictor (independent) variables are independent, it dramatically simplifies the task of classification, since it allows the class conditional densities $p(x_k \mid C_j)$ to be calculated separately for each variable, i.e., it reduces a multidimensional task to a number of one-dimensional ones. Furthermore, the assumption does not seem to greatly affect the posterior probabilities, especially in regions near decision boundaries, thus, leaving the classification task unaffected. A variety of methods exist for modeling the conditional distributions of the inputs. One choice is a normal distribution

$$p(x_k \mid C_j) = \frac{1}{\sigma_{kj}\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_{kj})^2}{2\sigma_{kj}}\right), \quad -\infty < x < \infty, -\infty < \mu_{kj} < \infty, \sigma_{kj} > 0$$

where μ_{kj} is the mean and σ_{kj} is the standard deviation. A second is a lognormal distribution

$$p(x_k \mid C_j) = \frac{1}{x\sigma_{kj}\sqrt{2\pi}} \exp\left(-\frac{[\log(x/m_{kj})]^2}{2\sigma_{kj}{}^2}\right), \quad -\infty < x < \infty, m_{kj} > 0, \sigma_{kj} > 0$$

where m_{kj} is a scale parameter and σ_{kj} is a shape parameter. A third possibility is a gamma distribution

$$p(x_k \mid C_j) = \frac{(x/b_{kj})^{c_{kj}-1}}{b_{kj}\Gamma(c_{kj})} \exp\left(-\frac{x}{b_{kj}}\right), \quad 0 < x < \infty, b_{kj} > 0, c_{kj} > 0$$

where b_{kj} is a scale parameter and c_{kj} is a shape parameter. A fourth possibility is a Poisson distribution

$$p(x_k \mid C_j) = \frac{\lambda_{kj}}{x!}\exp(-\lambda_{kj}), \quad 0 < x < \infty, \lambda_{kj} > 0, x = 0,1,2,\ldots$$

where λ_{kj} is the mean.

7.2 Training

The Statistics Toolbox (Version 6.1) provided by MathWorks as part of the Matlab™ (Release 2007b) computational environment [8] was used to apply a naive Bayes classifier to the multi-sensor data. The "classify" function within the Statistics Toolbox was employed using discriminant functions for naïve Bayes classifiers. The function was applied twice to check on sensitivity with regard to the choice of discriminant function. It was applied once by fitting with a multivariate normal density to each group with a pooled diagonal covariance matrix estimate ("diaglinear" choice for the type of discriminant function), and once by fitting with estimates stratified by group ("quadratic" choice for discriminant function). The feature vectors and class values for the training dataset were supplied to the function, and a prediction obtained for the feature vectors of the testing dataset in the same manner as for the CART model above. Equal probabilities were assigned as the prior probabilities for the groups, i.e. a uniform distribution.

7.3 Testing

The naïve Bayes classifier was tested by using the "classify" function to make predictions with the feature vectors from the testing dataset. The number of successful predictions with respect to the total number of feature vectors for each class was computed as a percentage, and the results obtained are shown and discussed in Chapter 9.

Chapter 8

Principal Component Regression

8.1 Method

The Principal Component Regression (PCR) method combines the Principal Component Analysis (PCA) decomposition with an Ordinary Least Squares (OLS) regression method to create a quantitative model for complex samples [21]. It is designed to confront the common situation where there are many (possibly correlated) predictor variables and relatively few samples. The basic goal is to project the observations (samples) X from a high-dimensional variable space to a low-dimensional subspace Y spanned by several linear combinations of the original variables. The projection subspace is then employed for the regression of Y.

PCA is used to generate a set of orthogonal principal components of X, from which the first k components are chosen for the projection subspace. The first k principal components correspond to the largest k eigenvalues and are constructed independently of Y. The value of k is chosen such that it that explains as much variance as possible in the independent variables (X) and is determined by leave-one-out cross validation. Restricting attention to principal components with the largest eigenvalues helps to control variance inflation but can introduce high bias by discarding components with small eigenvalues that may be most associated with Y.

In the usual multiple linear regression (MLR) context, the OLS solution for

$$Y = XB + A$$

is given by

$$B = (X^T X)^{-1} X^T Y \,.$$

The problem is that $X^T X$ is often singular, either because the number of variables (columns) in X exceeds the number of objects (rows), or because of co-linearities. PCR circumvents this by approximating X by the first k principal components, usually obtained from singular value decomposition (SVD)

$$X = X_{(k)} + A_{(k)} = (U_{(k)} D_{(k)}) V_{(k)}^T + A_{(k)},$$

decomposing X into orthogonal scores T and loadings P

$$X = T_{(k)} P_{(k)}^T + A_{(k)},$$

and regressing Y not on X itself but on the first k columns of the scores T. The scores are given by the left singular vectors of X, multiplied with the corresponding singular values, and the loadings are the right singular vectors of X. This leads to regression coefficients

$$B = P(T^T T)^{-1} T^T Y = VD^{-1} U^T Y$$

where the subscripts k have been dropped out of convenience.

PCR is founded on the premise that the independent variables are non-stochastic, i.e. there is no (or at least negligible) error in the independent variables. The performance of PCA in classification may not be satisfactory from the predictive point of view, because there is no guarantee that the principal component representing the large variance in X should necessarily be the component strongly related to dependent variables (Y).

When applying the PCR method for classification, the dependent variables for the training set must be expressed in a categorical form, e.g. class number. This is accomplished by creating a dummy variable for Y with values set to 1 if the sample X is in the class and 0 if it is not. The model, of course, will not predict either a 1 or 0 perfectly, so a limit must be set, say 0.5, above which the sample is estimated as a 1 and below which it is estimated as a 0. For multiple classes, a Y-block is created with each column corresponding to a variable for each class and a multivariate PCR is performed. For example, for a categorical dependent variable consisting of three classes (A, B, C), a Y-block may look like $Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, indicating the first sample belongs to class A, the second to class C, and the third to class B. After the multivariate regression is applied, the columns of the predicted Y-block are estimated as 0 or 1 as described above and the class identification subsequently made.

8.2 Training

The PLS_Toolbox (Version 4.2) developed by Eigenvector Research Inc. for multivariate analysis of chemometrics within the Matlab™ computational environment [8] was used to construct a PCR classification model. To set up the model for classification, the class values of the training dataset were first organized into a N×4 Y-block with values of 0 or 1, where the first column indicated the feature vectors in the N×10 float array belonging to class 1, the second column to class 2, etc. A value of 1 was used to identify the given feature vector as belonging to the class, and a value of 0 if it was not a member of the class. The training and testing datasets were then calibrated by centering the columns of the feature vectors to zero mean and scaling them to unit variance by using the "preprocess" function within PLS_Toolbox with the "calibrate" and "autoscale" options.

8.3 Validation

A cross validation was performed on the calibrated feature vectors and dummy categorical variables of the training dataset to determine the number of principal components to be used in the model construction. The "crossval" function was run for "pcr" with a leave one subset out ("loo") cross validation method and a maximum of 10 subsets. The variation in root mean square error of cross validation with principal component number was then examined for each of the four classes (Figure 10). The results show that for classes 2 and 4 the error reaches a minimum value after 5 principal components, while for classes 1 and 3 it is achieved after 7 principal components. Clearly, a choice of 7 principal components is sufficient to capture the maximum variance in the multi-channel sensor data. The PCR model was next constructed using the “pcr” function, with the calibrated training dataset as input, and autoscaling applied to the input feature vectors.

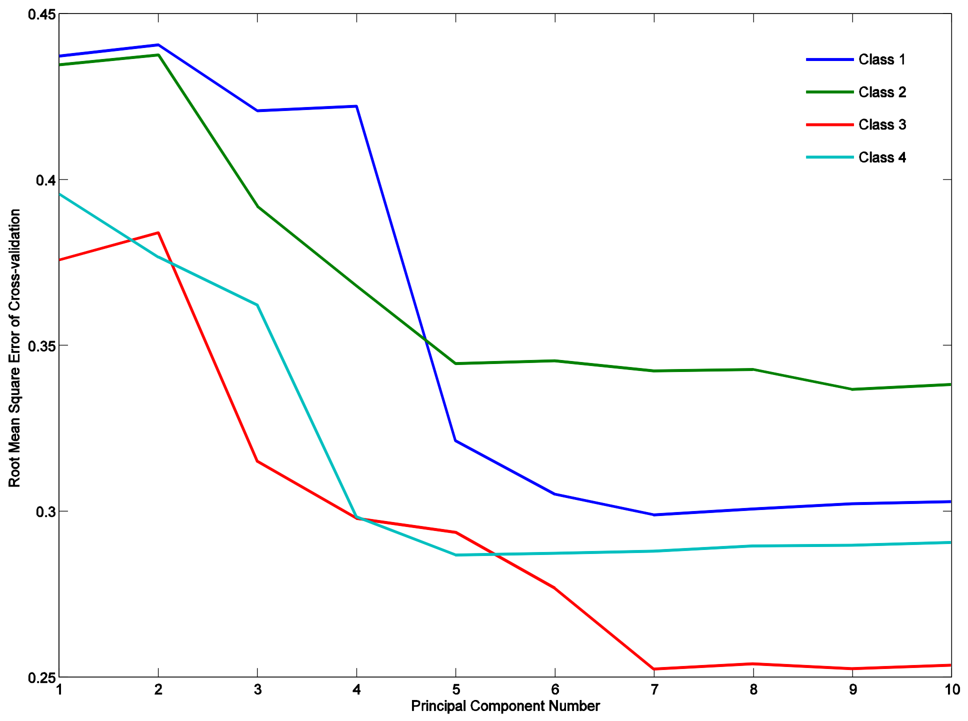


Figure 10. Variation of cross validation error with number of principal components for the 4 classes.

8.4 Testing

The PCR classifier was tested by using the “pcr” function to make predictions with the testing dataset. The calibrated feature vectors from the testing dataset were input along with the model constructed from the training dataset, with the raw residuals from all the predictors and predicted variables included in performing the classification. The number of successful predictions with respect to the total number of feature vectors for each class was computed as a percentage, and the results obtained are shown and discussed in Chapter 9.

Chapter 9

Results and Discussion

Six supervised machine learning techniques have been utilized to predict the class identification for an unknown dataset. The unknown dataset contains four subsets for each of the four classes. Each class subset has 14 unknown samples. The techniques include K-Nearest Neighbor (KNN), Support Vector Machines (SVM), Classification and Regression Tree (CART), Random Forest (RF), Principal Component Regression (PCR), and the Naïve Bayes Classifier. As described above, these techniques perform classification based on different prediction mechanisms. Below, we will first discuss the individual prediction result generated using each of these different techniques, and then make a comparison based on all the results produced by these techniques.

9.1 KNN

The prediction results for each class are listed in Table 6. The prediction success rate for each class and the overall success rate are shown in Figure 11. They reveal that a 100% success rate was obtained for class 4. Prediction of one out of 14 data samples failed for classes 1 and 3, and two out of 14 failed for class 2. Since the class of an unknown sample was assigned by determining the class of their nearest known neighbor ($k = 1$), it is important to understand how the feature vectors for each class are clustered, and how different classes are separated. To illustrate the relationship between the data samples, a dendrogram was generated for the training dataset.

When constructing a dendrogram, all of the Euclidean distances between samples are calculated and the samples with the smallest distance are found and linked together. The procedure is repeated and the samples with the next closest distance are found and linked. The results are then displayed as a connection dendrogram. Figure 12 shows a dendrogram generated for the training dataset using a single linkage algorithm with an unweighted average distance method (the Matlab “linkage” function with “average” chosen for method). The vertical bars indicate which samples or classes are linked, while the horizontal position of the bar indicates the distance between the linked samples or classes. Since there are 136 samples in the training dataset and it is not possible to display all individual samples along the vertical bar, we grouped those that have shorter distances to a cluster and display them using a node number, e.g. 33, 52, and 53 as shown in the figure. To easily locate the samples of different classes, we labeled the vertical bar with four colors that represent different classes: green for class1, yellow for class 2, blue for class 3, and red for class 4. From Figure 12, it is evident that samples of class 4 group into two clusters which are illustrated by the two red bars. This indicates that classification for class 4 could have a higher success rate. There are 14 out of 34 samples in class 3 that form two clusters that are separated from the others. The remaining samples in class 3 together with samples in both classes 1 and 2 form multiple clusters. Some of these clusters contain samples from single classes and others have samples from different classes. Also the distance between these clusters are relatively short, implying that classification for classes 1, 2, and 3 can have a higher failure rate than that for class 4.

As mentioned in Chapter 2, data samples within each class were collected from four different trials under different operational conditions, for the purpose of collecting a robust dataset. The sensor behavior may be dependent on the operational conditions and thus the data samples collected from different trials may have noticeable variations. This could be the reason for the separation of different clusters within a single class.

Table 6: Prediction results for the testing dataset using KNN method.

KNN	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	13	1	0	0	92.9
P - Class 2	1	12	1	0	
P - Class 3	0	0	13	0	
P - Class 4	0	1	0	14	

T - Class: Truth class ID

P - Class: Predicted class ID

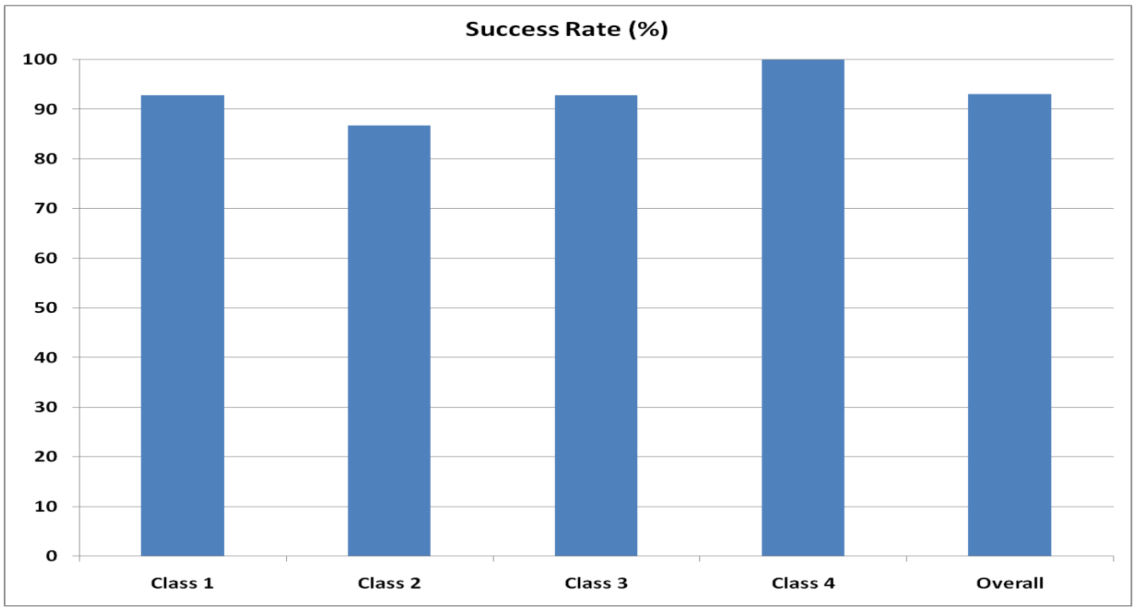


Figure 11. Prediction success rates for the testing dataset using KNN method.

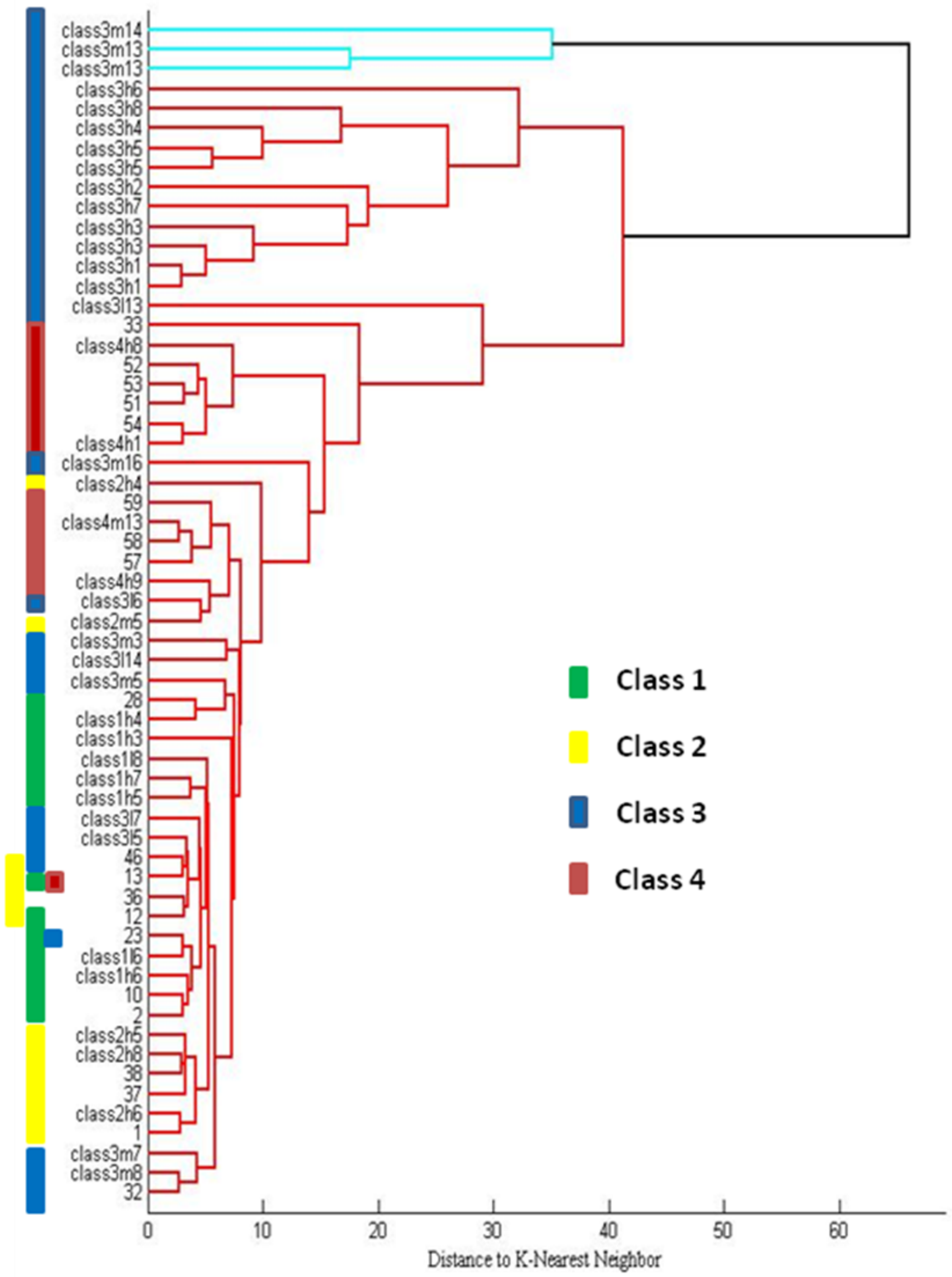


Figure 12. Dendrogram of the training dataset.

9.2 SVM

Similarly, a dataset containing 14x4 samples for four classes was used as an unknown dataset for class prediction using the SVM method. The prediction results are listed in Table 7. The success rates for each class are summarized in Figure 13. Class 4 has a 100% prediction success rate while the other classes have ~ 80% success rates. Cross-reactivity is observed between classes 1, 2, and 3.

As illustrated in the dendrogram shown in Figure 12, some samples for different classes in the training dataset group into a cluster, and multiple cases can be observed. The SVM method carries out the classification by creating separating hyperplanes to distinguish between objects of different class memberships. The cross-reactivity between different classes, as observed by the distance between different samples or groups, makes it difficult to create good hyperplanes that are able to separate the different classes. As a consequence, prediction for some samples in classes 1, 2, and 3 failed.

As is well known, one of the SVM drawbacks is that it is sensitive to noisy data. Classes 1, 2 and 3 in the dataset are noisy so that SVM is not able to find optimal hyperplanes for a good separation among these classes. Therefore the ~20% failure rates for classes 1, 2, and 3 are understandable.

SVM	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	12	1	0	0	87.5
P - Class 2	1	11	2	0	
P - Class 3	1	2	12	0	
P - Class 4	0	0	0	14	

T - Class: Truth class ID

P - Class: Predicted class ID

Table 7: Prediction results for the testing dataset using SVM method.

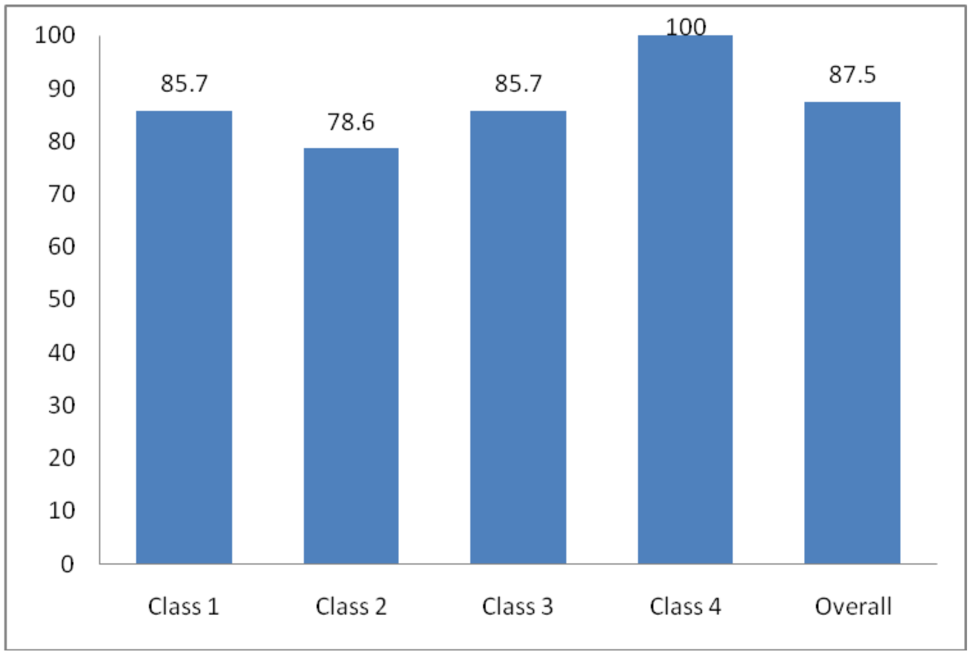


Figure 13. Prediction success rates for the testing dataset using SVM method.

9.3 CART

Table 8 provides the prediction results for the testing dataset using CART method. The prediction success rates are shown in Figure 14. The overall prediction success rate is quite good (92.9%). The prediction success rates are 100% for classes 3 and 4, 85.7% (2 out of 14 testing samples) for both classes 1 and 2. The misclassification occurred between classes 1 and 2. The following discussion may explain these prediction results.

As shown in Figure 9, the decision tree built using the training dataset has only one end node for class 4, indicating that class 4 is well separated from the others. The 100% prediction success rate is therefore not surprising. The amplitude of the sensor responses from channels 3, 5, and 9 are quite distinct and strong for class 3 (Figure 3). As shown in the tree, most class 3 end nodes were generated using sensor responses from either channels 3, 5, or 9, and as a consequence, class 3 is well classified. It is notable that 2 samples in each class 1 and class 2 are misclassified as class 4. This indicates that prediction of classes 1 and 2 could have cross reactivity with class 4. Indeed, as shown in Figure 3 (predictor values vs. predictor number), features of samples with low predictor values in classes 1 and 2 look similar to those in class 4. Therefore, such cross-reactivity may be understandable

The test results suggest that CART is flexible in practice in that it can easily model nonlinear or nonsmooth relationships. It has the ability to interpret interactions among predictions, and has great interpretability due to its binary structure. Typically, another decision-tree based classification method (Random Forest) is used to improve upon these predictions. This will be discussed below.

Table 8: Prediction results for the testing dataset using CART method.

CART	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	12	0	0	0	92.9
P - Class 2	0	12	0	0	
P - Class 3	0	0	14	0	
P - Class 4	2	2	0	14	

T - Class: Truth class ID

P - Class: Predicted class ID

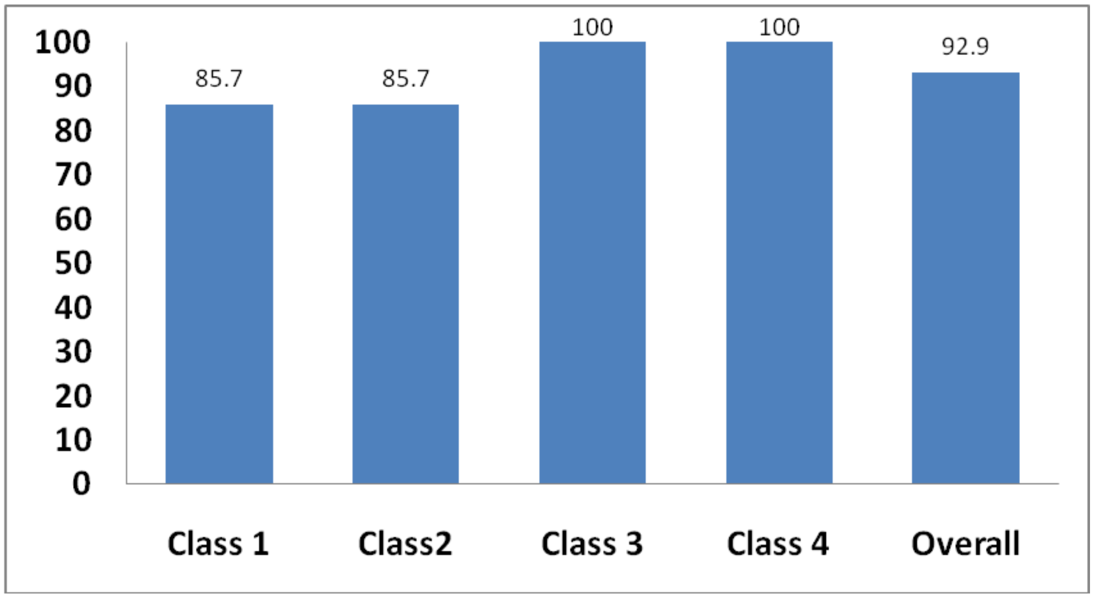


Figure 14. Classification success rates for the testing dataset using CART method.

9.4 Random Forest

The Random Forest method was used to predict the testing dataset. The latter contains a total of 56 samples, 14 samples for each class. Different numbers of trees were used for classification to explore how the success rate is affected by the number of trees. Table 9 lists an example of the success rates for various numbers of trees, and Table 10 provides an example of prediction results using 50 trees in the forest. With a 91% overall prediction success rate, it can be said the result is quite good. It reveals the prediction success rate can be improved using multiple trees (≥ 5). It was also observed the prediction for classes 1, 3, and 4 were always 100% when the number of trees used to build the forest reached 5. However, any further increase in number of trees did not have an obvious impact on the prediction success rate, although a slight increase in success rate was observed when the number of trees reached 40. Since the results generated using CART are already quite good for this particular dataset, the benefits gained in prediction by using a RF are not compelling.

It is believed the Random Forest can improve the success rate of CART since it uses multiple trees for prediction. This has been clearly demonstrated here since a single decision tree (c.f. CART) can handle this particular dataset well. The prediction result listed in Table 9 is a result from a single run of the random forest function. Results from different runs are change because the process of building the forest is dependent upon a distribution of random numbers. This could be a drawback of random forests due to the lack of reproducibility.

Table 9. Dependence of classification success rate on the number of trees used in Random Forest method*.

Number of Trees	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Overall**
1	85.7	78.6	100	100	91.07
5	100	78.6	100	100	94.65
10	100	78.6	100	100	94.65
15	100	78.6	100	100	94.65
20	100	78.6	100	100	94.65
25	100	78.6	100	100	94.65
30	100	78.6	100	100	94.65
40	100	85.7	100	100	96.42
50	100	85.7	100	100	96.42
60	100	78.6	100	100	96.42
70	100	85.7	100	100	96.42

* Total 56 testing data samples, 14 for each class, were used for prediction.

Table 10: Prediction results for the testing dataset using RF method with 50 trees.

RF	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	14	1	0	0	96.4
P - Class 2	0	12	0	0	
P - Class 3	0	0	14	0	
P - Class 4	0	1	0	14	

T - Class: Truth class ID

P - Class: Predicted class ID

9.5 Naïve Bayes Classifier

The Naïve Bayes method was applied to classify the testing dataset. Both linear and quadratic disciminant analyses were used to make predictions. The linear discriminant analysis uses hyperplanes for decision boundaries while the quadratic discriminant analysis exploits hyperellipses. Results obtained using both discriminant analyses are shown in Table 11 and Figure 15. There is no obvious difference between the results generated using linear and quadratic discriminant analyses. Apparently, the overall prediction success rates are lower, indicating the Naïve Bayes method may not suitable for this sensor data.

Naïve Bayes invokes the simple assumption that independent variables are statistically independent, that is there are no correlations between the feature values of different sensor channels. To perform the classification, it determines the posterior probability of a test sample being a member of a particular class by calculating the probability of a single feature value of the sample becoming a member of the same group of the feature values from the class. This assumed simplicity is inappropriate for the data generated from the chemical sensor array we are studying here.

All chem./bio sensor arrays are designed and implemented in such way that different channels respond to a single stimuli in various ways, rather than each channel responds to a different stimuli. As a consequence certain correlations exist among either a subset or all of the channels. Moreover, these correlations should be the key components exploited for classification. Since the Naïve Bayes method ignores such correlations it is not a suitable classifier for our dataset.

Table 11: Prediction results for the testing dataset using Naive Bayes method.

NB-Linear	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	12	0	0	0	76.8
P - Class 2	2	13	5	5	
P - Class 3	0	0	9	0	
P - Class 4	0	1	0	9	
NB-Quardratic	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	11	0	2	1	76.8
P - Class 2	3	12	0	4	
P - Class 3	0	0	11	0	
P - Class 4	0	2	1	9	

T - Class: Truth class ID

P - Class: Predicted class ID

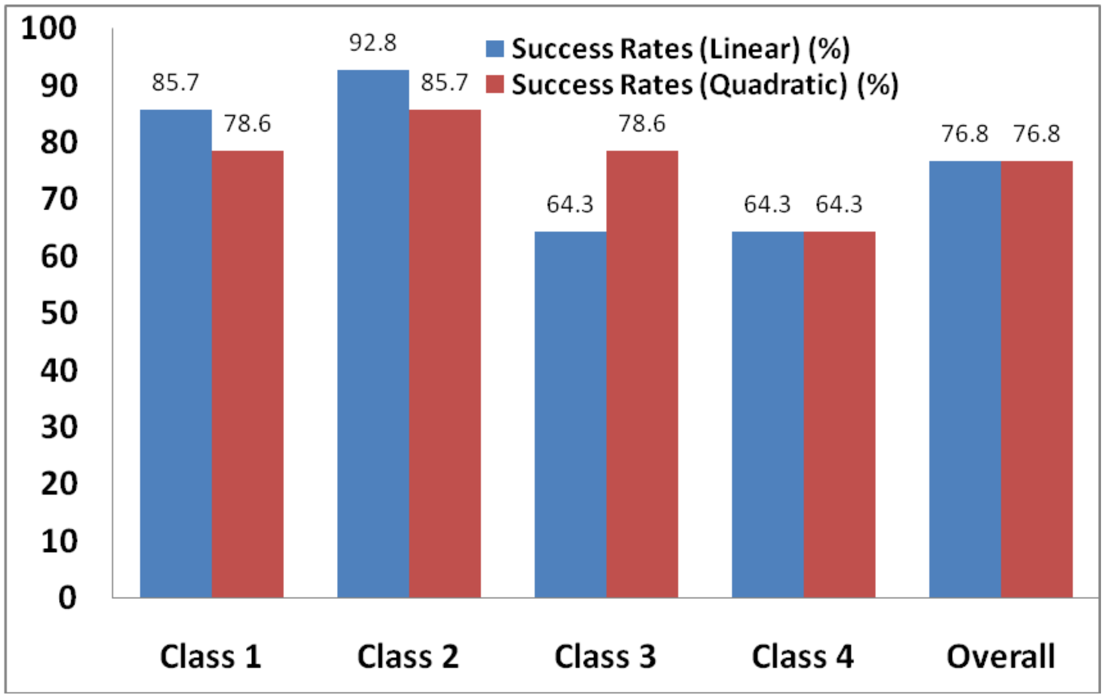


Figure 15. Classification success rates using Naïve Bayes Classifer with both linear and quadratic disciminant analyses.

9.6 PCR

Based on the PCR model built using the training dataset, the regression coefficients were obtained using 7 principal components calculated via the Singular Value Decomposition (SVD) method. Predictions for the testing dataset were then made for the calibrated feature vectors of the testing dataset using these regression coefficients. Figure 16 shows the predicted Y-block values for each class. Using the typical criteria of $Y \geq 0.5$ (a member) and $Y < 0.5$ (not a member), one can assign a class identification to the testing data samples [22].

As shown in Table 12, 9 out of 14 samples were correctly identified for classes 1, 2, and 3. 8 out of 14 samples were successfully identified for class 4. It was noticed that all misclassification are false negatives rather than false positives. For those testing samples that were classified (35 samples), the prediction success rate is 100%. However, the overall success rate is only 62%. The remaining 21 failure samples were not classified as a member of any class. Attempts have been made to reduce the criteria Y value (0.5) to a lower value (e.g.0.4) to try to obtain a higher yield. However, a higher cross-reactivity was observed between different classes.

From the above discussion, one concludes that using the typical criteria of $Y \geq 0.5$ (a member) and $Y < 0.5$ (not a member) for classification may post a strong constraint for each unknown sample to be classified. Therefore, we propose using an alternative approach for real-time classification that can satisfy the requirement of online processing. Every time an unknown sample is run through the training model, a set of Y-block values is generated for predicting membership in each class. For example, in this study, there are 4 Y-block values predicting membership in each of the 4 classes. Instead of using the threshold criteria $Y \geq 0.5$, the class associated with the maximum value in the Y-block set is used to assign the class of an unknown sample as illustrated in Table 13. Using this approach a class can be assigned to all the unknown samples. The prediction results for the testing dataset are shown in Table 14 and the success rates for each class are provided in Figure 17. The success rates for classes 2 and 4 are quite good, but those for class 4 are relatively low. It also found that almost all the cross-reactivity occurs for samples collected from low concentration challenges. As discussed in Chapter 2, cross-reactivity was observed for feature vectors of data samples collected from low concentration challenges, and this may be responsible for the PCR results.

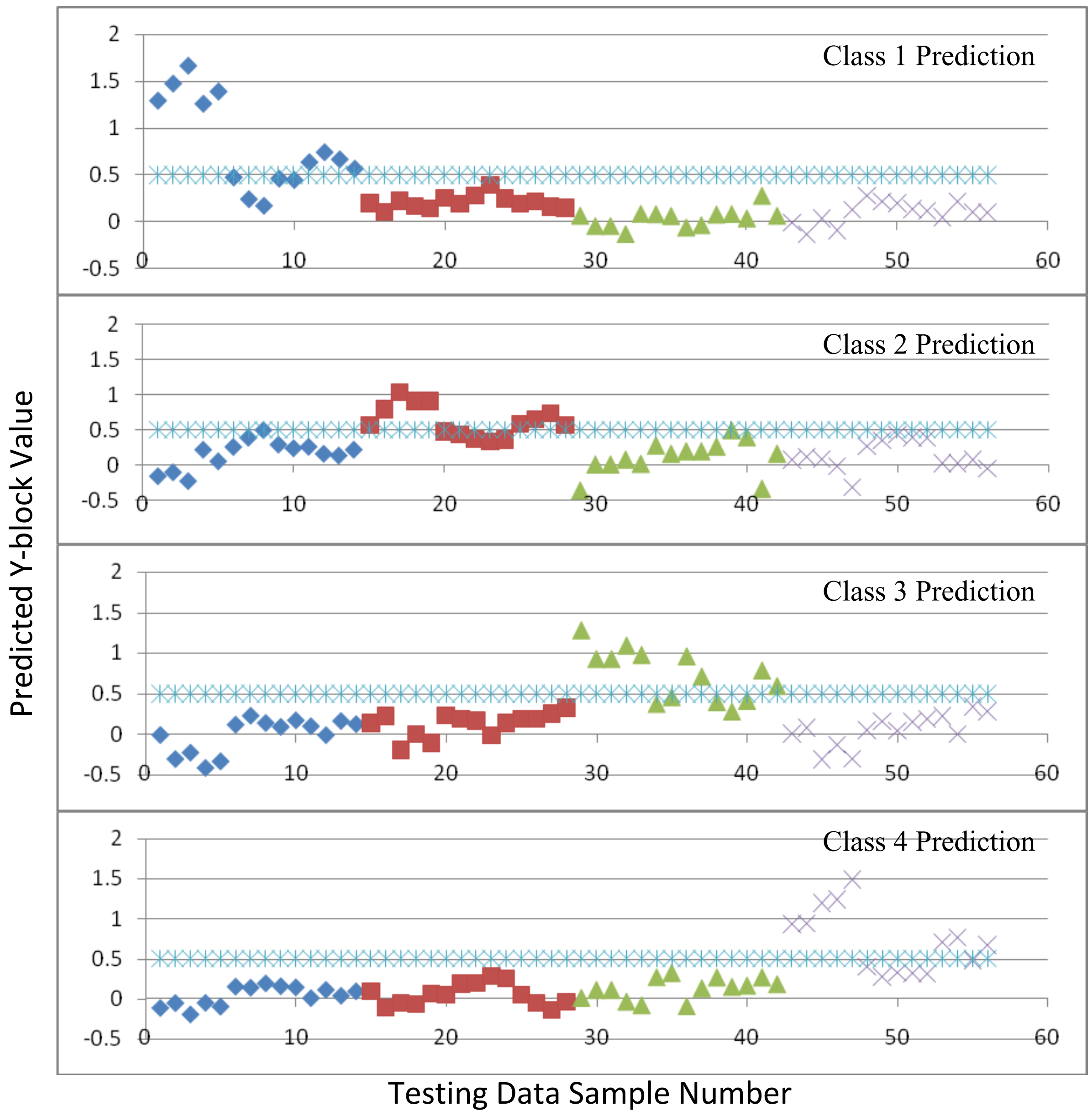


Figure 16. Predicted Y-block values for each class.

A Y-block is a N×M matrix, where N is the number of samples in each class and M is the number of classes. For training data, each Y-block value is assigned a value of 1 if it is a class member and 0 otherwise. For predicted results, the values of the Y-block typically fall within the range 0 to 1.

Truth data:
Sample number 1 – 14: Class 1 ◆
Sample number 15 – 28: Class 2 ■
Sample number 29 – 42: Class 3 ▲
Sample number 43 – 56: Class 4 ×

Table 12: Prediction results for the testing dataset using PCR method.

PCR	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)	Overall (%)
P - Class 1	9	0	0	0	100.0	62.5
P - Class 2	0	9	0	0		
P - Class 3	0	0	9	0		
P - Class 4	0	0	0	8		
Not classified	5	5	5	6		

T - Class: Truth class ID

P - Class: Predicted class ID

Table 13: Classification of unknown samples based on their maximum Y-block value.

Sample #	P-Y Block 1	P-Y Block 2	P-Y Block 3	P-Y Block 4	Determined Class	Truth Class
1	***1.2901***	-0.1598	-0.0110	-0.1194	***1***	1
2	***1.4715***	-0.1040	-0.3101	-0.0574	***1***	1
3	***1.6611***	-0.2302	-0.2301	-0.2007	***1***	1
4	***1.2578***	0.2172	-0.4206	-0.0544	***1***	1
5	***1.3879***	0.0525	-0.3387	-0.1017	***1***	1
6	***0.4767***	0.2574	0.1168	0.1492	***1***	1
7	0.2452	***0.3902***	0.2282	0.1364	***2***	1
8	0.1750	***0.4971***	0.1381	0.1898	***2***	1
…						
21	0.1940	***0.4298***	0.1868	0.1894	***2***	2
22	0.2818	***0.3581***	0.1656	0.1944	***2***	2
23	***0.3951***	0.3307	-0.0115	0.2857	***1***	2
24	0.2511	***0.3501***	0.1366	0.2622	***2***	2
25	0.1898	***0.5773***	0.1875	0.0455	***2***	2
…						
38	0.0813	0.2583	***0.3945***	0.2659	***3***	3
39	0.0848	***0.4866***	0.2784	0.1502	***2***	3
40	0.0390	0.3877	***0.4089***	0.1644	***3***	3
41	0.2812	-0.3349	***0.7851***	0.2686	***3***	3
42	0.0663	0.1588	***0.5932***	0.1817	***3***	3
…						
47	0.1318	-0.3188	-0.3037	***1.4907***	***4***	4
48	0.2785	0.2696	0.0531	***0.3988***	***4***	4
49	0.2212	***0.3496***	0.1590	0.2703	***2***	4
50	0.2004	***0.4308***	0.0424	0.3264	***2***	4
51	0.1373	***0.3908***	0.1535	0.3184	***2***	4

Note: P-Y Block n: predicted Y Block value for class n (n = 1, 2, 3, 4)

This is the maximum predicted Y-block value for unknown sample 51. Since it is the predicted Y-block value for class 2 (P-Y Block 2), the unknown sample is assigned as a member of class 2.

Table 14: Prediction results for the testing dataset based on maximum Y-block value from PCR method.

PCR	T - Class 1	T - Class 2	T - Class 3	T - Class 4	Overall (%)
P - Class 1	12	1	0	0	85.7
P - Class 2	2	13	1	4	
P - Class 3	0	0	13	0	
P - Class 4	0	0	0	10	

T - Class: Truth class ID
P - Class: Predicted class ID

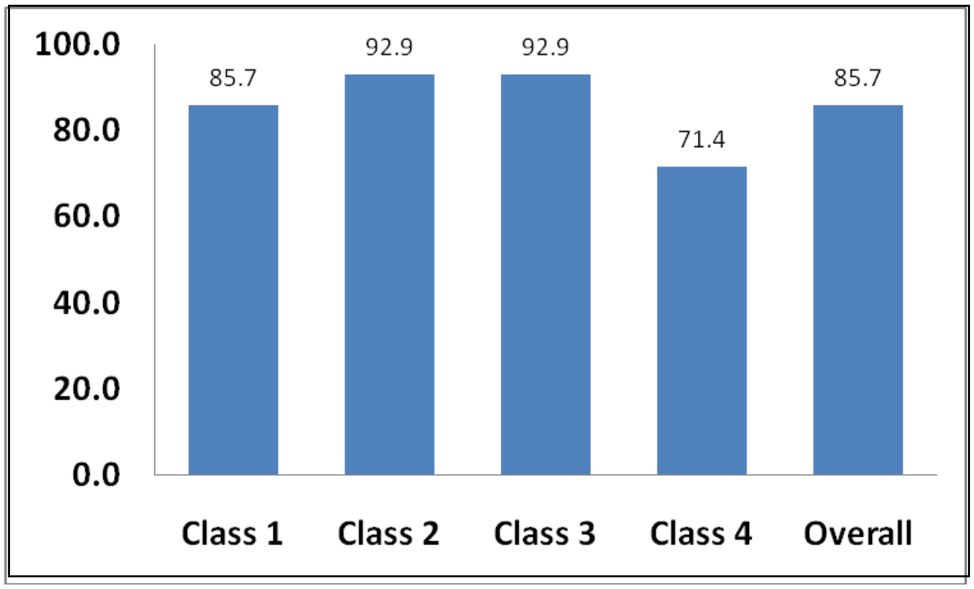


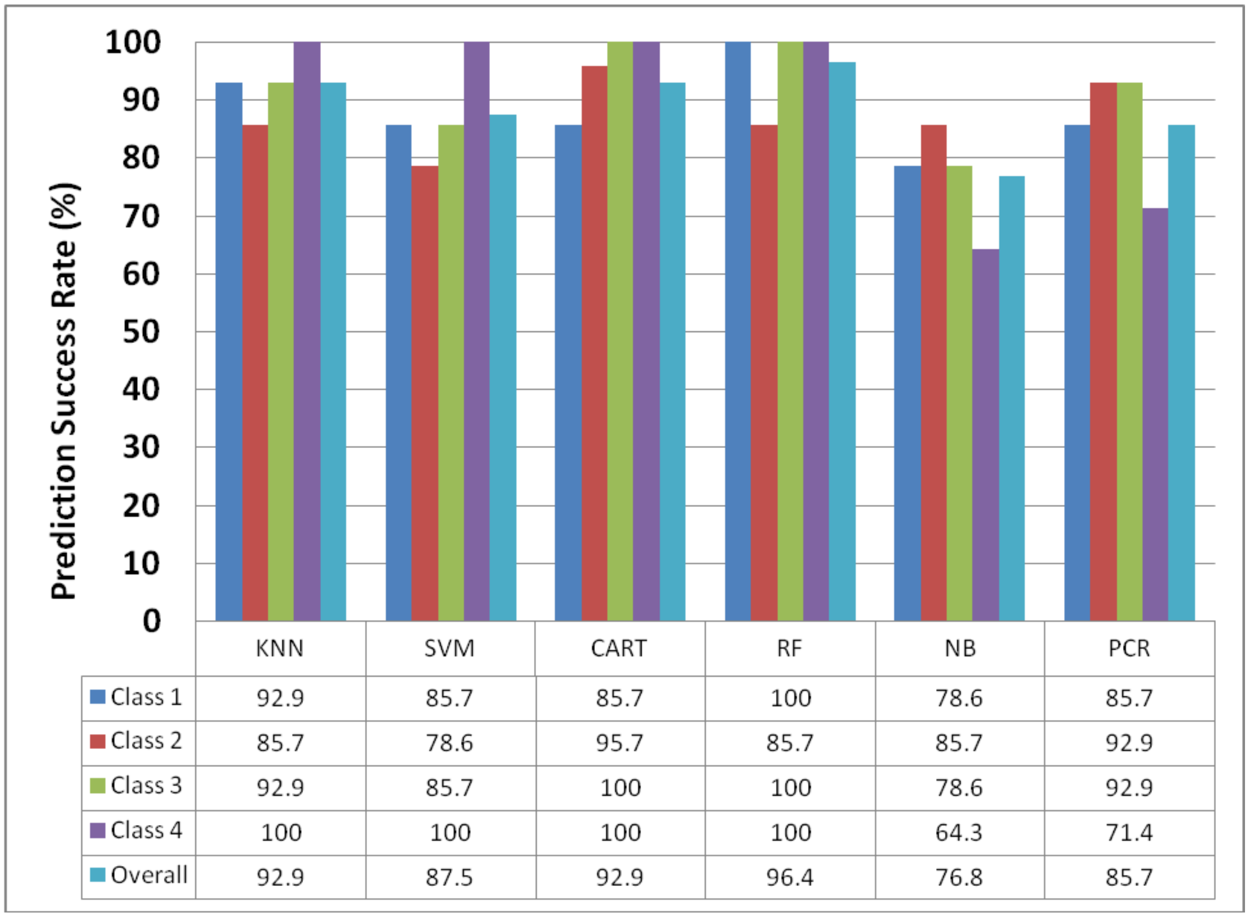
Figure 17: Prediction success rates for the testing dataset based on maximum Y-Block value from PCR method.

9.7 Comparison

A comparison was made based on the prediction success rates of each method for different classes and the average success rates across all classes. A comparison of the classification performance of these 6 methods is schematically shown in Figure 18. Apparently, the tree-based methods CART and RF, along with the distance-discriminating KNN method, have the best performance with an overall success rate above 90%. CART and RF discriminate data samples by directly using the feature values of the feature vector. The KNN method performs discrimination based on the distance between samples in a 10-dimensional (10 predictors) space. Although KNN doesn't directly utilize the feature values for classification, it calculates the distances directly using the feature values of the feature vector. These methods are not based on any assumptions that directly conflict with the sensor behavior.

The NB, PCR, and SVM models have a relatively poor performance than the CART, RF, and KNN models with an overall success rate of 70 to 80%. This was caused by a variety of reasons for the different methods. As described earlier, the assumption of independence of variables in NB, the cross-reactivity between the feature vectors of data collected from low concentration chemical challenges in PCR, and the noisy nature of the dataset for SVM, are the reasons for the failures.

For the sensor detection algorithm, one important factor that needs to be considered is the required computing time for execution of the model because a rapid response is essential. Both KNN and CART are fast, while RF and PCR require more time. The NB and SVM methods need the longest time to produce predictions.



	KNN	SVM	CART	RF	NB	PCR
🟦 Class 1	92.9	85.7	85.7	100	78.6	85.7
🟥 Class 2	85.7	78.6	95.7	85.7	85.7	92.9
🟩 Class 3	92.9	85.7	100	100	78.6	92.9
🟪 Class 4	100	100	100	100	64.3	71.4
🟦 Overall	92.9	87.5	92.9	96.4	76.8	85.7

Figure 18: Comparison of prediction success rates of six methods.

Chapter 10

Conclusion

In this study, we have explored applying six supervised machine learning techniques to classify a dataset collected using a chemical sensor array. The six machine learning methods include distance-discriminating K-Nearest Neighbor (KNN) and Support Vector machine (SVM), tree-based Classification and Regression Trees (CART) and random Forest (RF), Bayesian-type Naïve Bayes Classifier (NB), and the regression-based Principal Component Regression (PCR).

The multi-channel sensor dataset used for this study was collected by challenging a sensor array with 4 groups of chemicals with different molecular structures. Each data sample is a combined response from 10 different types of chemical sensors (channels) on a sensor array. Each group of chemicals was tested for three concentration levels (high, medium, and low), and the data collected from challenges with the same group of chemicals form a class. A feature vector was extracted for each sample, and is composed of 10 predictors, where each predictor is associated with one of the 10 sensor channels. The training dataset for constructing the classifiers has 4 classes of 34 data samples, and the testing dataset has 4 classes of 14 data samples.

The results obtained from applying the machine learning methods showed that the tree-based methods CART and RF, along with the distance-discriminating method KNN, yield the best performance with an overall success rate above 90%. Although the RF outperforms all other classifiers it requires a longer computing time. KNN and CART produced good results and the computation is quick. The NB, PCR, and SVM have relatively poor performance with an overall success rate ranging from 71% to 85%. This was caused by a variety of reasons for the different methods. The success rate using PCR was the lowest (~60%) if a threshold criteria (Y-block value ≥ 0.5) is used to determine the class membership of the unknown sample. This poor performance can be improved if the maximum Y-Block value is used to assign the class membership of the unknown sample.

The outcome of this study suggests that machine learning techniques can be exploited to classify data generated using the multi-channel chemical sensor array. Due to the nature of the dataset, RF and CART, along with KNN would be the best candidate classifiers. Future work could focus on continuing the testing using a larger dataset, and using a dataset collected under different operational conditions to improve the robustness of the classification algorithm.

References

[1] W. Gopel, Chemical imaging: I. Concepts and Visions for Electronic and Bioelectronic Noses, Sensors and Actuators B 52 (1998) 125-142.

[2] ERI, PLS_Toolbox, http://www.eigenvector.com/

[3] SAW MiniCAD mk II™ Miniature Chemical Agent Detector, http://www.msanorthamerica.com/catalog/product500650.html

[4] Cyranose 320 E-nose User Manual, Smiths Detection, User Manual,

[5] M. Barker and W. Rayens, Partial Least Squares for Discrimination, J. Chemometrics 17 (2003) 166-173.

[6] G. Musumarra, V. Barresi, D. F. Condorelli, C. G. Fortuna and S. Scirè, Potentialities of Multivariate Approaches in Genome-Based Cancer Research: Identification of Candidate Genes for New Diagnostics by PLS Discriminant Analysis, J. Chemometrics 18 (2004) 125-132.

[7] T. M. Cover and P. E. Hart, Nearest Neighbor Pattern Classification, IEEE Transactions on Information Theory 13 (1967) 21-27.

[8] The MathWorks, http://www.mathworks.com/

[9] N. Cristianini and J. Shawe-Taylor, An Introduction to Support Vector Machines: and Other Kernel-based Learning Methods, Cambridge University Press, Cambridge, 2000, p. 189.

[10] V. N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998, p. 732

[11] K. Pelckmans and J. Suykens, LS-SVM Lab, http://www.esat.kuleuven.ac.be/sista/lssvmlab

[12] K. Pelckmans, J. A. K. Suykens, T. Van Gestel, J. De Brabanter, L. Lukas, B. Hamers, B. De Moor and J. Vandewalle, *LS-SVMlab: a MATLAB/C toolbox for Least Squares Support Vector Machines*, 2003, p. 8.

[13] K. Pelckmans, J. A. K. Suykens, T. Van Gestel, J. De Brabanter, L. Lukas, B. Hamers, B. De Moor and J. Vandewalle, LS-SVMlab Toolbox User's Guide, version 1.5, ESAT-SCD-SISTA K.U. Leuven, Leuven-Heverlee, Belgium, 2003, p. 106

[14] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, Classification and Regression Trees, Wadsworth International Group, Belmont, CA, 1984, p. 358.

[15] G. Biau, L. Devroye and G. Lugosi, Consistency of Random Forests and Other Averaging Classifiers, The Journal of Machine Learning Research 9 (2008) 2015-2033.

[16] L. Breiman, Random Forests, Machine Learning 45 (2001) 5-32.

[17] L. Breiman and A. Cutler, Random Forests, http://www.stat.berkeley.edu/users/breiman/RandomForests

[18] X.-W. Chen and J. C. Jeong, Sequence-based Prediction of Protein Interaction Sites with an Integrative Method, Bioinformatics 25 (2009) 585 - 591.

[19] X.-W. Chen and J. C. Jeong, Sequence-based Prediction of Protein Interaction Sites with an Integrative Method, http://www.ittc.ku.edu/~xwchen/bindingsite/prediction.htm

[20] G. H. John and P. Langley, Estimating Continuous Distributions in Bayesian Classifiers, in: International Conference on Uncertainty in Artificial Intelligence, San Mateo, CA, 1995, pp. 338-345.

[21] H. Wold, *Estimation of Principal Components and Related Models by Iterative Least Squares*, in Multivariate Analysis (Ed. P. R. Krishnaiah), Academic Press, New York, 1966, pp. 391-420.
[22] B. M. Wise, N. B. Gallagher, R. Bro, J. M. Shaver, W. Windig and R. S. Koch, PLS_Toolbox 4.0 for use with MATLAB™, Eigenvector Research Inc., Wenatchee, WA, 2006, p. 414

Vita

Dr. Sichu Li received a B.Sc. degree in Chemistry in 1985 and a M.Sc. degree in Polymer Chemistry in 1988 from Zhongshan University in China. She received her Ph.D. in Material Chemistry in 1997 from the Advanced Material Research Institute at the University of New Orleans. After completing her Ph.D. she worked in academia for 5 years developing novel nanostructured materials for various applications. She then joined Smiths Detection at Boston (formerly Echo Technologies) in 2003, where she directed a team to conduct multiple projects developing sensors for real-time detection and classification of airborne chemical and biological agents. Dr. Li joined MITRE Corporation at Washington in 2008, where she serves as a lead scientist continuing her efforts in developing sensors for chem/bio detection.

Dr. Li has published more than 30 scientific papers in peer-reviewed journals, and holds two pending patents, one for chem/bio sensor system design and one for sensor detection algorithm.